Indaleko

Towards More Useful File Systems

by

William Anthony Mason

S.B. Mathematics, University of Chicago, 1987

MSc. Computer Science, Georgia Institute of Technology, 2017

A THESIS PROPOSAL SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

The Faculty of Science

(Computer Science)

The University of British Columbia

(Vancouver)

August 2021

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Indaleko

submitted by **William Anthony Mason** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Computer Science**.

Examining Committee:

Margo I. Seltzer, Computer Science
Co-Supervisor

Ada Gavrilovska, Georgia Institute of Technology, College of Computing
Co-Supervisor

Sasha Fedorova, Electrical and Computer Engineering
Supervisory Committee Member

Norman Hutchinson, Computer Science
Supervisory Committee Member

Andrew Warfield, Computer Science
Supervisory Committee Member

TBD
TBD

Abstract

Modern storage systems have evolved from a simple model in which names denoted the information necessary to know both *where* a file was stored as well as *what* that file represented. Today, users routinely store information in a variety of different distinct storage silos, each of which has its own naming scheme, characteristics, and meta-data support. Storage silos can consist of storage local to the computer, shared with other local computers, stored in geo-replicated distributed storage systems, focused more on object storage than traditional hierarchical file organization, and communicated through non-traditional storage systems as part of collaborative efforts. There is little to assist those using the myriad of separate storage silos keep track of the relationships between them.

What this means is a user trying to locate a document she saved while collaborating with a colleague will need to search through each of these storage silos that she routinely uses: was the document attached as an e-mail, sent via a collaboration service, stored in a shared cloud storage location will have a difficult time locating that document six months later.

Revision: September 10, 2021

Contents

List of Tables

List of Figures

Glossary

This glossary uses the handy `acroynym` package to automatically maintain the glossary. It uses the package's `printonlyused` option to include only those acronyms explicitly referenced in the LaTeX source. To change how the acronyms are rendered, change the `\acsfont` definition in `diss.tex`.

HCI Human-Computer Interface

NVMe Non-Volatile Memory Express

Acknowledgments

Thank those people who helped you.

Don't forget your parents or loved ones.

You may wish to acknowledge your funding sources.

Chapter 1

Motivation and Problem Statement

> *For every particular thing to have a name is impossible. — First, it is beyond the power of human capacity to frame and retain distinct ideas of all the particular things we meet with: every bird and beast men saw; every tree and plant that affected the senses, could not find a place in the most capacious understanding.* — **John Locke**, *An Essay Concerning Human Understanding* [1]

Naming is an essential human task. We use names to describe identity, relationship, and property. Thus, naming is quite broad, as it can be quite concrete — *identity* being particularly concrete — to quite abstract concepts such as *similarity*.

Computer storage primarily focuses on *identity*, with some limited flexibility to encode relationships and properties. It has been "good enough" to keep us suffering, but not good enough to have enjoyed decades of criticisms and attempts to find a better model.

One early model, Memex, described how computers can help human users by serving as a form of "augmented memory" describes a system in which properties and relationships are used to make humans more productive [2]. Despite stunning progress in many aspects of computer storage systems in the past 70 years, those storage systems have failed to improve on naming and organization in any meaningful way that permits realizing this plausible vision of how information should be organized.

1.1 Thesis Statement

To address the demands to organize, locate, and manage the rapidly growing body of data collected, stored, and organized by modern computer systems we must rethink the existing naming model of current computer storage systems and implement a richer, dynamic, and functional naming scheme capable of satisfying significantly more of those needs.

1.2 Why this is a Computer Systems problem

> *Paradigm paralysis refers to the refusal or inability to think or see outside or beyond the current framework or way of thinking or seeing or perceiving things. Paradigm paralysis is often used to indicate a general lack of cognitive flexibility and adaptability of thinking.* — The Oxford Review Encyclopedia of Terms (2021).

One challenge in developing this work is the resistance of the computer systems community to considering *naming* as a systems problem. One common response is to posit that this is an Human-Computer Interface (HCI) problem. While there are certainly aspects of data visualization that *are* HCI problems, a review of the current state of affairs suggests this is clearly untrue.

- The computer systems community has *already* taken over at least some aspect of naming; it is dereliction of responsibility to now insist the current state of affairs is not tightly tied to earlier choices by the computer systems community.
- The HCI community has been evaluating, reviewing, and proposing potential alternatives to the current computer storage naming paradigm for *decades* but these solutions fall short of being viable because it is not sufficient to change a single application — even something as core to the problem as the file browser — to resolve this problem.
- The computer storage community does admit to the challenges here and have implemented system-specific solutions to improving naming, but human users do not live in a reality in which such narrow solutions resolve the naming challenges of the larger system — it is not sufficient to argue that any storage system has solved this problem when users are called upon to use multiple storage systems on a daily basis.

The framework of modern file systems differs little from the framework used in Multics in 1965, itself based upon the familiar file cabinet metaphor described for electronic storage and data organization in the 1950s. The technological components that are used to construct file systems have changed dramatically in the ensuing decades, as storage devices have become smaller, higher density, more capable and less expensive. In 1965 the total amount of data storage was less than the capacity of a single Non-Volatile Memory Express (NVMe) storage device, yet the *framework* for how we organize, present, managed, and manipulate data has not changed. If anything, additional aspects of how we build file systems have ossified into common models: tight integration with the operating system, a presumption of block structured storage and memory page management.

While the HCI community is likely to contribute novel new ways of presenting information to users, they cannot do so until the systems community offers them the tools necessary to do so.

1.3 Challenges

TM ►*I say quite a few things here, which I need to back up with suitable references.*◄
TM ►*I have mulled over whether this belongs here or in the Background/Related work section.*◄

~~The computer systems community discourages challenging the current file system naming model: what we have works, it is not the purview of the systems community to address issues around utility, and our research is properly focused on supporting innovative new hardware solutions and optimizing their performance. *In spite* of this,~~ reality indicates that our storage models, which has already been evolving, is showing further signs of change. The space between file systems, which provide a basic abstraction of unmanaged data, and database systems, which provide a basic abstraction of managed data, is filled with a growing array of *semi-structured* mechanisms including: key-value stores, object stores, documents stores, no-SQL databases, data warehouses, and data lakes.

Each new mechanism that we invent for storing data creates a new "silo" of that information, often with its own semantics and meta-data, and seldom with any explicit way of tying related information together across such silos. Indeed, silos create the fundamental challenge for which I am proposing research to understand and evaluate potential solutions. Storage silos are often designed with specific usage patterns in mind: Intel DAOS is an HPC

platform that is part of their "exascale" storage stack [1].

However, the commonly used mechanism for naming — the hierarchical namespace — confounds the storage *location* with the information *relationship*. In other words, ordinary applications familiar with using hierarchical file systems expect related objects, of whatever type, to be in or close to the same directory. The hierarchical file system model has been highly successful for more than a half-century, though that success is from the perspective of the storage community. The Human-Computer Interface (HCI) community has been pointing out that this model is deficient for many users.

Silos are not a new problem. The systems community has addressed them via "mount points". All mainstream consumer operating systems at present (Linux, MacOS X, and Windows) support mount points. NFS on UNIX is implemented using explicit mount points, much like media file system instances while AFS used a global namespace mechanism to connect its silos ("volumes") together so that users were given a single consistent namespace. More recent innovation has created internet-based namespaces, such as Amazon's S3, which is an object store. Some of those web namespaces can be mounted as file systems, such as using Web Distributed Authoring and Versioning (WebDAV), which converts the internet `GET/PUT` model into an hierarchical file systems namespace model. There are numerous "S3" FUSE file system implementations on github.com, not because they seek to be efficient but because that hierarchical model is the one understood by existing applications.

These solutions are inadequate: they still conflate *location* with *context*. Some of this is historical: the only reliable place where applications can store context is within the file name. Using file names to embed context ("meta-data") is well-understood [3]. Yet, embedding context in file names does not solve the problem of storing dissimilar types of data in the same location, which defeats the purpose of having specialized storage. Similarly, it does not solve the problem of placing data objects in their optimal storage location while preserving their relationship.

TM ▶*I'm not sure I like this example, but I'll leave it here for the time being.*◀ The need for this is increasingly clear. For example, Qumulo has focused on improving the manageability of large data collections: "When people are dealing with petabytes of storage and billions of files and they're using scaleable storage, they tend to run into problems not with storage itself but

[1] https://www.intel.com/content/www/us/en/high-performance-computing/daos-high-performance-storage-brief.html

the data"[2]. While Qumulo's approach is motivated by the same problem, their solution emphasizes scale and manageability within a new meta-silo, which does not address the core problem of organizing naming in a silo-independent fashion.

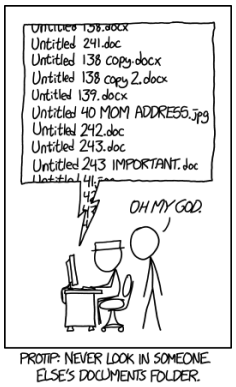

Figure 1.1: XKCD: Never Look in Someone Else's Documents Folder

Media file systems have long been organized using a simple internal key-value store. The BSD UNIX Fast File System used an "index node" (inode) as a description of a data object, such as a file or directory [4]. These nodes were indexed using an identifier, which acts as a simple key. The NTFS file system is structured similarly [5]. Recent work explored the idea of separating the namespace from storage with an emphasis on high performance solid-state storage (SSD) devices [6]. This echos earlier work suggesting using object storage devices (OSD) for file systems implementations [7].

Beyond separating naming from storage, there is also a trend to separate meta-data from storage as well. One early example of this is Lustre [8], with more recent work including Ceph and Gluster [9], [10].

The logic of separating meta-data from data solves one of the problems of storage, namely that the logical location of information can be distinguished from the physical location of data, which has been previously explored and deemed broadly beneficial. [11] This approach logically makes sense when considered in the multi-silo context as well, since a human user looking for something actually does not care *where* that the thing is located they care *what* the thing is.

This raises an intriguing question, one that is at the heart of my own research: *what is the purpose of naming?* Applications are quite happy to use

[2] https://www.seattletimes.com/business/storage-startup-qumulo-opens-the-kimono/

names that are meaningless to humans, such as content hashes, UUIDs, or randomly generated string names, such as are used for temporary application files. Indeed, if an application knows that a name is a fixed size, this can be used to simplify the implementation or to separate application named files from other files. An application that uses UUID names can limit its field of interest quite quickly by simply restricting itself to the 36 character format commonly used of 36 hexadecimal digits with four '-' separator characters breaking the UUID into five parts.

Humans use file names to provide *context* as to what is in the named object. Librarians assisting people in organizing data routinely suggest embedding contextual information in the file names; directories are then used to create additional organizational structure. There are numerous limitations to this approach, however, because humans do not organize things in the same fashion and even the same human does not organize things the same over time. This is such a common phenomenon even the popular press pokes fun at it (Figure 1.1.)

The inability to find specific things is *not* unique to computer data storage. Indeed the current situation for organizing digital data seems to be descended from the organizational structure of a library or a file cabinet, despite the fact the physical limitations on files, which in turn leads to a similar result: a large collection of files, spread over multiple different locations, each with its own organizational structure makes it difficult for the original creator of that content to find a specific file, let alone an outsider attempting to find relevant information.

Indeed, the system we have evolved works *in spite* of the obvious artificial limitations imposed by decades old decisions. Rather than being a deep insight that the current system works because it is the best of all possible models, it is a testament to human ingenuity and a tolerance for primitive, sub-standard systems.

There are a body of recommendations about file naming standards[3] [4] [5] [6]. Harvard Data Management suggests [7]:

- Think about your files

[3] Data Management for Researchers [12]

[4] Smithsonian: https://library.si.edu/sites/default/files/tutorial/pdf/filenamingorganizing20180227.pdf

[5] Stanford: https://library.stanford.edu/research/data-management-services/data-best-practices/best-practices-file-naming

[6] NIST: https://www.nist.gov/system/files/documents/pml/wmd/labmetrology/ElectronicFileOrganizationTips-2016-03.pdf

[7] https://datamanagement.hms.harvard.edu/collect/file-naming-conventions

- Identify metadata (e.g., date, sample, experiment)
- Abbreviate or encode metadata
- Use versioning
- Think about how you will search for your files
- Deliberately separate metadata elements
- Write down your naming conventions

Thus, there are common themes: a name provides context for *what* the given thing represents. Uniformity of information is also important — the "naming convention" permits not only identifying similarity but key elements of *difference* between any two named things. Thus, to return to the original question: "what is the purpose of naming?"

TM ▶*It seems that this Harvard list is a serious indictment of the existing system: it pushes the cognitive load onto the users, talks about meta-data , versioning, encoding,* and *capturing the naming convention.*◀

It seems clear that human naming is not the same as data storage naming, though we often treat them as the same. This can be seen in the data storage tendency to either use names that are entirely about locating a specific object, such as is the case with a key-value store, for example, or it combines location with identity, such as is typified by the Uniform Resource Identifier (URI) [13].

Thus, it would seem that naming is about:

Identity — the name of a thing should be sufficient to verify that it is the specific thing we seek. An example of this is biometric data for humans or a cryptographic hash for a storage object;

Location — the name of a thing could provide information that directly or indirectly specifies where it is located. A human example is how an identity card might specify a current residential address. A data storage example is the U.S. Library of Congress Classification System, which can be used to find a particular work within very large bodies of work even though books are typically not thought of as a storage media;

Relationships — the concept that one thing can be derived from another, whether in a direct form, such as a version of the same logical thing, or in a more indirect form, such as a provenance relationship showing

that one thing was derived from another thing by applying some set of transformations to it;

Characteristics — this relates to something that is a property of the thing. For humans this might be the date of birth ("creation date"), height, weight, eye color, etc. For storage objects this might include timestamps, size, data type;

Context — at some some level, we often rely upon names to provide us with *context*. For humans, we assume that people with the same family name are likely to be members of the same family, even though "Aki Smith" is distinguished from "Dagon Smith".

When considered from this perspective, it seems clear why "naming is hard" — it plays multiple distinct roles, sometimes overlapping, sometimes interfering. These challenges are not well-served by existing naming support in computer storage systems.

TM ►*I feel that I should capture the dynamic nature of naming, which really does differ from the traditional static model of it.* ◄

1.4 Contributions

> *[I don't let the cleaners in b]ecause **I** know where everything in this room is. All the books, the papers — and the moment they start cleaning, those things get hopelessly organized and tucked away and I can never find them again.* — Crown of Midnight, Sarah J. Maas

This thesis proposal describes what I intend to contribute to our understanding of how to evolve naming in computer storage:

1. An explicit model for what computer storage naming is: how and why we name data objects, what a name *should be* in a computer storage context to correspond how names are used and formed; and

2. An architecture and design of a potential naming system that supports the naming model (item 1 that is sufficiently flexible to work across the full range of current and prospective computer storage needs: in-memory compute systems, small devices, computer workstations, enterprise scale storage systems, and distributed, geo-replicated cloud storage systems, which permits the construction of new purpose-built

storage systems with the strong naming support as well as integration of existing storage systems; and

3. Implement and evaluate a novel storage system implementation based on this design (item 2) that demonstrate strong naming support within a single storage silo model; and

4. Implement and evaluate an implementation of the richer naming system (item 2) on an existing computer storage system; and

5. Implement and evaluate the combination of both the novel storage system implementation (item 3) and the enhanced legacy system (item 4) into a unified system consistent with the model (item 2).

Chapter 2

Related Work

The optimal way to organize data within a storage silo has been extensively studied from multiple perspectives. That there are so many different approaches to evaluating the optimal way to organize things is a testament to both the importance and complexity involved in approaching this topic. This section will consider the following related work:

- The *storage* perspective, which is primarily rooted in the systems perspective. Storage in this context includes file systems and databases, each of which then has multiple different manifestations.

- The *provenance* perspective, which is related to systems but focuses on creating a context of explainability that is distinct from storage.

- The *human-computer interface* (HCI) perspective, which is related to how humans organize and find information within storage systems. There are a number of distinct perspectives to this work including: hiearchical data organization, personal information management, enhanced search, and cognitive data organization.

Each of these perspectives is complementary and assist in better understanding the problem: systems focuses on being able to efficiently and reliably store and recover information though often it is agnostic to the specifics of the information; provenance focuses on accountability and explainability; and HCI focuses on the human facing problems that need to be solved and tends to ignore how those solutions are implemented.

[3], [14]–[46]

2.1 Storage

Much of storage work is dominated by the realities of how media and networks function, with a goal towards increasing both capacity and performance. The basic model of data organization within storage systems tends to separate into *structured* data — typically what is found in databases — and *unstrutured* data — typically what is found in file systems or object stores.

TM ►*Describe structured data: why do people use databases, what are the strengths and weaknesses. How does this relate to data organiztion?* ◄

TM ►*Describe unstructured data: why do people use file systems or object stores, what are the strengths and weaknesses? How does this relate to data organization?* ◄

TM ►*Describe the work that has been done in semantic file systems, metadata augmentation, and non-hierarchical organization structure (e.g., Ground and Placeless).* ◄

2.2 Provenance

TM ►*Explain provenance. It is a more recent area of study, but it also captures more of the kinds of information that will be useful to me.* ◄

2.3 Human-Computer Interface

The traditional primary organizational model provided to users has been the file/folder metaphor TM ►*Need citation to 1965 Daley and the 1956 storage papers*◄, which was itself derived from physical filing cabinets. However, electronic storage is not bound by the same restrictions as a physical filing cabinet, as can be seen even in early work TM ►*Again, this is Daley*◄ where the model added *links*; the closest equivalent to this in filing cabinets would be to make duplicate copies of a document — I have seen exactly this routinely practiced by bookkeeping and accounting professional, whether using physical or electronic filing systems. This works because the objects are generally *immutable*, but for electronic storage systems without deduplication it is not particularly space efficient.

The Human-Computer Interface community has been studying human-centered data organization models for decades. For example, the HCI community observed that hierarchical file structure is challenging for users with low spatial abilities [47]. This suggests why storage developers would not even see there is a problem here: the study of computer science correlates well with the development of spatial abilities [48]. In my own discussions with even senior computer scientists it is the introduction of *silos* that seems to

make the hierarchical abstraction break. "Did I store that in Dropbox, or Google Drive, or was it on my laptop or my desktop computer?"

The wealth of research here is astonishing, yet does not appear to influence the design of storage systems to exploit the results of that research. Indeed, the systems community seems to be focused on a *search based* solution while the HCI community research suggests that search is *not* preferred by users — instead, users want to *navigate*:

> *When retrieving a file the user needs to choose between folder based navigation and query based search. There are obvious intuitive advantages of search for both retrieval and organization. Search seems to be more flexible and efficient for retrieval. It is flexible because it does not depend on users remembering the correct storage location; instead, in their query users can specify any file attribute they happen to remember...*
>
> *In fact, regardless of search engine quality, people consistently use search only as a 'last resort' for that minority of cases where they cannot remember file locations...* [49]

Thus, from an HCI perspective it would seem that one potential research direction would be to consider potential interfaces that mimic navigation over a search interface.

For example, recent work around data curation explores the idea of a "data dashboard" since the first step of curation is finding specific data to curate [50]. This work builds upon prior research showing that when presenting data to users it is important to ignore storage silo boundaries. Indeed, my reading of this work is that it presents what seems to be navigation even though it is implemented using a search mechanism. Equally important, this work also points to the benefits of not changing the *location* of data, but rather allowing construction of useful relationships via metadata. Thus, this work supports my ideas of ignoring silo boundaries and providing metadata for use by a similar tool. What it does not explore is a way for data storage to provide enhanced metadata, dynamic update, and notification of changes, which are important elements to making such a dashboard more useful for data visualization.

Chapter 3

Model

In Section 1.3 I attempted to capture key aspects of how naming is used by human users and from that proposed a basic list of key elements to consider in a comprehensive naming model: identity, location, relationship, characteristic, and context. While this is a good place to begin my analysis more is needed to construct a robust model. For example, one of the challenges of ~~aspects of~~ naming is that they are *dynamic*: the storage location of a given data object might change. The *object* is not different but its storage location is: users are unlikely to care about this specific detail until they go to actually retrieve it and find out that location is inaccessible.

To facilitate developing my naming model, I rely upon several use-cases that have arisen during the course of my research around this topic, both working with collaborators as well as discussions for ordinary users that are unfamiliar with my research area.

3.1 Use Cases

To motivate the model I propose for *Indaleko*, I first start with a series of potential use cases. TM ▶*Note that I've started with the two from the original* Indaleko *paper, but I think it might be worthwhile to add one or two more to provide a more well-rounded model.*◀

Data Processing

Compliance

Memex

Asset Management

Feature	Existing Technologies	No Solution
ACTIVITY CONTEXT	timestamps and geo-location, image recognition, browsing history, ticketing systems, application-specific solutions like Burrito [3].	Link related activity across apps, record browsing history and chat conversations relevant to the creation of the data object, storing it in ways that are secure and compact.
CROSS-SILO SEARCH	Search by name, creator, content across silos, app-specific searches (e.g., Spotlight)	Unified search across all kinds of storage, including file systems, object stores, apps and devices
DATA RELATIONSHIPS	De-duplication of documents, versioning of specific files, git ancestor relation	Explicit notion of data identity, tracking different versions across different silos as data is transformed
NOTIFICATIONS	File watchers (INotify), synchronization status, manually inspecting modified time	Ability to subscribe to specific changes on attributes
PERSONALIZED NAMESPACE	Hierarchy plus hard/soft links. Use of tags.	Creating personalized namespaces with with flexible data organization and views

Table 3.1: Use-case driven functional requirements.

Using these uses cases as motivation, I propose that *Indaleko* support the features as shown in Table 3.1 in greater detail in Section 3.2.

3.1.1 Data Processing

TM ►*This is from the HotStorage paper submission*◄

Aki and Fenix are preparing a report summarizing their work on a data analysis project for a customer. Fenix sends an email to Aki containing a CSV file with original data. Aki opens this document in Excel, formats and filters it, adds additional data from a corporate storage silo, and then returns the Excel document to Fenix on Slack. Fenix is away from their desk when it arrives, so they open it on their phone, uploading it to a cloud drive. Fenix then sends the link to Aki for editing with update notifications. Finally,

Fenix sends a PDF of the report to the compliance officer who promptly asks, "Where did this data come from?" ~~Aki and Fenix are preparing a report summarizing their work on a data analysis project for a customer. Fenix sends an email to Aki containing a CSV file with original data. Aki opens this document in Excel, formats and filters it, adds additional data from a corporate storage silo, and then returns the Excel document to Fenix on Slack. Fenix is away from their desk when it arrives, so they open it on their phone, uploading it to a cloud drive. Fenix then sends the link to Aki for editing with update notifications. Finally, Fenix sends a PDF of the report to the compliance officer who promptly asks, "Where did this data come from?"~~

3.1.2 Compliance

Delete Request: Some time later, the compliance officer requests that all documents containing a customer's data must be deleted. To help with finding all relevant customer data, Dagon joins the project and examines the report and requests the original data from which it was produced. Aki remembers that they gave the original data to Fenix shortly after collecting it, but does not remember the name, location, or even how the relevant files were transmitted. Thus, Aki has to manually search possible locations and applications, sendsing references to documents to Dagon, who then starts organizing these files to methodically identify the ones that might contain the customer's data. In the process, many of the other team members' references to the documents stop working.

TM ▶ *Discussion with Ada: how do people do this already? Why are those solutions insufficient? Some sites are accessible and others are not. How do they prove they are GDPR compliant? What about when people move from dynamic to static memory? Could I use storing the hash value as a mechanism for motivating this because it facilitates finding things. Much like the Apple content hash for child p0rn. How about extracting text from pictures to avoid censorship?* ◀

3.1.3 Memex

The "memex" is a device posited by Vannevar Bush in 1945 [2]:

> *"Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and, to coin one at random, "memex" will do. A memex is a device in which an individual stores all his books, records, and communications,*

and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory."

While the world wide web is certainly one interpretation of his forward thinking article, the system he describes is also highly personal and appears to focus not only on finding things but also capturing the context in which they are found.

TM ►*Seems like the key here is to extract the salient factors that would impact this model.* ◄

Thus, when considering my naming model, I can draw upon the needs of Memex to assist in ensuring the model is sufficient to meet the issues raised by Bush [2]:

Selection — "The prime action of use is selection, and here we are halting indeed." The key here is being able to *filter* items of interest at a given time. This becomes important as the number of objects being considered grows. While computers are somewhat faster than they were in 1945, he points out the general problem of scaling and the need to be able to limit the actual size of the search space. This is a very real consideration for our naming system: scalability. Brute force search is not sufficient, as that is what we have *now* and even as fast as storage systems are today this is not tenable.

Association — "Our ineptitude in getting at the record is largely caused by the artificiality of systems of indexing. When data of any sort are placed in storage, they are filed alphabetically or numerically, and information is found (when it is) by tracing it down from subclass to subclass. It can be in only one place, unless duplicates are used; one has to have rules as to which path will locate it, and the rules are cumbersome. Having found one item, moreover, one has to emerge from the system and re-enter on a new path.

"The human mind does not work that way. It operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain. It has other characteristics, of course; trails that are not frequently followed are prone to fade, items are not fully permanent, memory is transitory. Yet the speed of action, the intricacy of trails, the detail of mental pictures, is awe-inspiring beyond all else in nature."

Thus, the key take-away here is to find associations. This is part of the motivation for *activity context*.

Trails — "And his trails do not fade. Several years later, his talk with a friend turns to the queer ways in which a people resist innovations, even of vital interest. He has an example, in the fact that the outraged Europeans still failed to adopt the Turkish bow. In fact he has a trail on it. A touch brings up the code book. Tapping a few keys projects the head of the trail."

Thus, the key take-away here is to be able to show at least one kind of relationship: a "trail," which seems to be similar to provenance.

3.1.4 Asset Management

One recurring theme in my conversations with the creative community has revolved around the management of *assets*. This term is broadly used: web pages are constructed of assets, video sequences consist of a unique rendering of independently combined asets, computer games include audio and video assets. There is no single hierarchical organizational structure for assets that satisfies the needs of this type of creative endeavor: a single game asset could be classified by a myriad of characteristics. When a game developer is looking for a particular asset, they are often focused on those characteristics. When an audio engineer is attempting to construct specific sound effects they could be looking for: the duration, number of channels, channel mapping, sampling frequency, bit depth, instrument, or dynamic range for example. It is clear that what doesn't work in such a situation is a single directory filled with all of the assets: that isn't useful.

Indeed, this use case seems to focus on being able to identify and use the *properties* of a given data object. Simpler examples of this might be how one organizes documents for accounting purposes: bank statements could be sorted by the bank from which they came or the month that they cover. In my experience, accounting users actually will store copies of such a file because they need to be able to identify it in *both* formats. Another similar example is "how do you organize your music collection?" A music collection could be organized by artist, or album, or year of release, or publisher or lyricist, or genre. When we have a physical copy of recorded music (e.g., an eight-track tape [1] we are limited to the ways in which we can organize

[1] Invented in 1964 and thus as old as Multics, which adopted the hierarchical file system structure.

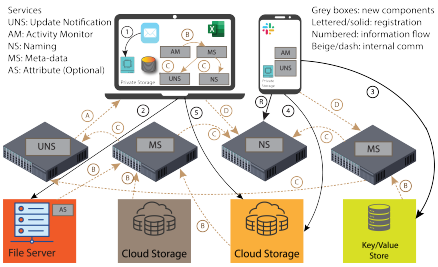

Figure 3.1: Indaleko Architecture (see Section 3.3.1).
Grey boxes indicate new components. AM="Activity Monitor", MS="Meta-data Server", UNS="Update Notification Server", NS="Namespace Server".

it. Digital files have no such limitations; that limitation is an artifact of the current naming system.

TM ▶*What requirements are imposed by this use case? It's relatable, but does it really impact the design?* ◀

3.2 Features

Indaleko must provide the following features to meet the needs of the use cases:

Activity Context — this is a mechanism by which we capture information for understanding the context in which data is created, transformed, and accessed. This is not application or storage silo specific information. Examples of this might include timestamps, application specific meta-data, history information, provenance information, etc.

Search — while search is not the only use we envision for *Indaleko*, it is one of the mechanisms that we expect to enable and should support searching across storage silos and exploiting the richer meta-data

3.3 Architecture

TM ▶*I'm starting with the* Kwishut *architecture from HotStorage 2021.*◀

Indaleko is a family of services that enable sophisticated search and naming capabilities. The key features that differentiate Indaleko from prior work are:

1) incorporating object relationships as first class meta-data because prior work has not done so which interfers with the ability to ensure meta-data and naming services work together; and

2) federating meta-data services, which is necessary to ensure efficiency, scalability, and security; and

3) recording activity context, which is necessary to provide insight into how the data is used dynamically (over its lifetime) rather than statically (at the point of creation or last update); and

4) integrating storage from multiple silos, which is necessary to clearly distinguish the storage characteristics that are focused on storage service optimization from the usage characteristics that establish associative context; and

5) enabling customizable naming services, which are needed because we need the flexibility to support a broad range of existing as well as innovative new storage services.

Data continues to reside in existing and to-be-developed storage silos. Indaleko interacts with these silos, collects and captures metadata, and provides a federated network of metadata and naming services to meet the needs of users with the use cases in §A.6.3 being our initial evaluation of our architecture and design.

3.3.1 Indaleko Services

Figure 3.1 illustrates the Indaleko architecture. Indaleko allows for different deployment scenarios. The services can be run independently, they can be co-located and bundled together to run on a local device, integrated into an OS, or available as network-based services.

In the discussion below, parenthesized numbers and letters refer to the arrows in Figure 3.1. There are five main components:

1) **Metadata servers (MS)** are responsible for storing attributes and provide a superset of capabilities found in existing metadata services [25], [39]. Users can register an MS with activity monitors or attribute services, which allows the MS to receive updated attributes from storage objects and activities (B). Thus, there can be multiple sources of attributes including the user itself. Metadata servers may retain the full or partial history of attribute updates or maintain only the most recent value.

2) **Namespace servers (NS)** connect to one or more MS and use the metadata to provide users with a personalized namespace that allows both manual organization (i.e., a hierarchical namespace) and rich search capabilities. Users can register with an NS (R) that uses one or more MS to obtain relevant attributes from them (C). Additionally, users can be part of a corporate NS that allows sharing of their select metadata with other users via standard enterprise public-key cryptography.

3) **Activity monitors (AM)** run on the user's devices. Their main function is to observe temporal relations, activity context, and relationships between objects on a user's device and transmit them to an MS (D).

4) **Attribute services (AS)** extract attributes from storage objects and transmit them to an MS (B). An AS might be invoked on updates, run once or periodically. For example, a file system AS would update the object's metadata with basic attributes such as size or modification time. There can be many AS that extract more "interesting" attributes, e.g., image recognition, similarity, or other classifiers.

5) **Update notification server (UNS)** provides notification mechanisms. Users can register interest in changes of attributes or underlying storage and will receive a message on change events (A) to which they have access.

3.3.2 Indaleko working example

To make the Indaleko architecture concrete, we revisit our use-cases from §A.6.3 and walk through parts of it to illustrate how Indaleko supports the various actions and events.

Storing the e-mail attachment. Aki's act of saving the CSV file that Fenix sent in email corresponds to the creation of a new object on the file server silo, i.e., the file system (4). The file server is Indaleko-aware, so the AS co-located with it extracts attributes from the document and forwards them to the MS (B).

The AM on Aki's laptop detects that the CSV file came via company email from Fenix. It then captures the activity context identifying the relationship between the e-mail and the CSV file and transmits it as additional metadata about the CSV file to the MS (that already contains metadata extracted by the AS). Moreover, because there is a company-wide namespace service, Indaleko establishes that the e-mail attachment, the CSV in the file

server, and the one on Fenix's laptop (from which the file was sent) are exact copies of each other.

Many applications already record some form of activity context, e.g., chat history, browsing history. Such histories provide a rich source of additional metadata. Other activity context, specifically the relationship between objects, such as the fact that a particular file was saved to a local storage device from an email message, requires more pervasive monitoring as found in, e.g., whole provenance capture systems [46]. Indaleko is agnostic about the precise data that comprises activity context, but allows for storing and accessing activity context as metadata.

Creating the Excel file. Next, Aki opens the CSV file using Excel and stores it as a spread sheet. This creates a new object. The AM detects that the newly created spreadsheet is a conversion from the CSV file, either via a notification from Indaleko-aware Excel or by monitoring the system calls executed on the local system. Aki proceeds to modify the data by filtering it in Excel and saving the changes. The AM records this event and updates the meta-data of the spreadsheet to record the derivation-relationship. Ideally a Indaleko-aware version of Excel specifies to the AM the exact type of the relationship (in this case a derivation); otherwise the AM informs the MS about an unspecified data relationship by observing the opening of a CSV file and a subsequent creation of the Excel file.

Sharing the spreadsheet. As Fenix receives the Excel file from Aki via Slack on their phone, a sequence of metadata events similar to those described earlier takes place, except the phone does not run a local NS or MS. Fenix now uploads the file to the company's cloud drive (4). The MS (by way of AS) reflects the creation of a new object and records its remote location. The use of a company-wide namespace and metadata service enables Indaleko to record that the file in the cloud drive is, in fact, a copy of the one received via Slack. Further, Fenix informs their personal NS that they wish to notify Aki about all updates to the file on the cloud drive. Thus, whenever an AS sends updated attributes to the MS, Fenix receives a notification.

Data origin and delete requests. When the compliance officer asks about the origin of the data, Fenix can query the corporate NS to obtain the complete history of the report. This includes the spreadsheet from which the report was derived and the e-mail or Slack messages that transmitted the files. The corporate NS was configured to be aware of the locations of the collaborating users' personal NS. Moreover, because of the activity contexts captured by the AM, Indaleko is able to identify documents that were created during any activity involving the customer whose data must

be deleted. Starting from these documents, and by using the relationship of documents, Dagon was able to find all relevant objects and delete them, including the e-mail and Slack messages.

Note that unlike existing systems, Indaleko is able to efficiently find related objects across storage silos. Operating systems already provide users with indexing services to accelerate search of local files. This search can be made cross-silo by mounting and enabling indexing on network shares (e.g., Windows Desktop Search), or by interfacing with specific applications such as e-mail (e.g., MacOS Spotlight, or Android search). The problems with indexing on a large network storage repository are resource limitations such as bandwidth and local storage that may render the system unusable during indexing. In contrast, Indaleko addresses these limitations by delegating indexing and storage to one or more services.

NS are responsible for providing efficient search functionality. Indaleko uses AS to keep attributes up to date with object modifications. Lastly, Indaleko supports coordinated search among one or more local and remote NS, allowing, for example, a user to search across both their local NS as well as their employer's NS.

TM ►*Would it be useful to resurrect some of the excluded text from this section?*◄

Chapter 4

Plan

The contributions that I proposed in Section 1.4 are substantial. To achieve this I must implement a number of novel new approaches to naming in computer storage systems. In this chapter I provide a high level overview of my plan to accomplish this.

4.1 Research Questions

This plan is focused on answering the following research questions:

Can a storage silo implementation that separates naming, meta-data, and storage manag
It is easy to talk about *functionality* but there is a real risk with a radically new architecture that it will not be usable due to the performance costs. Part of the challenge in answering this research question is to pick appropriate benchmarks. My goal is to demonstrate there is *no* significant cost associated with I/O performance (less than 5%), and that there is at most modest cost associated with meta-data performance (creating, opening, deleting, renaming) versus existing systems (less than 10%). While I think it would be useful to work with someone interested in data visualization to demonstrate improvements in data analysis and access, I do not propose doing this as part of my own work. I discuss this in more detail in Section 4.2.

Can a federated meta-data service be constructed to provide meaningful security guaran
By increasing the exposure of information via enhanced/augmented meta-data there are clear benefits for compliance measures, but these are achieved at the expense of making additional meta-data available. Part of the original design was to address these concerns and this

research question evaluates the efficacy of that design. To do this will require examining the security model my system uses, the potential information exposure, and at least a first-order analysis of the impact of information leaks. Depending upon my analysis of the answer to this question it may be necessary to augment the *Indaleko* security model to better protect the information.

How can *Indaleko*, implemented as a cross-silo naming and meta-data service, provide m

The research question itself is vague at this point because while I can suggest anticipated benefits in terms of cross-silo data location it is difficult to identify the "benefits" let alone quantify them. Despite this, both will be necessary, which makes this a "high risk" research question. TM ►*I don't think this is a good research question as written, so I need to re-work this…again. The difficulty here is that measuring this is not going to be particularly easy, unless we turn this into an HCI activity, such as by looking at "abandonment rates" when people start searching for things. That* may *be what is required, though, to make this argument because a performance eval of the software doesn't make much sense — to what would I compare this?*◄

At present, I only have a basic architecture and high level design of the system. While developing all of the pieces required to realize the design would take considerable effort, I can construct a prototype implementation using existing technology for many of the components. In the following sections, I have attempted to identify potential existing technologies that can be used. My goal in identifying existing technologies that I can use is sufficient to validate my goal of prototype construction while leaving sufficient flexibility to replace components as it proves necessary during the implementation phase of my research.

One key area in which I expect to develop new techniques is meta-data extraction and construction from existing data and storing that meta-data into pre-existing technology components. I describe this further in the following sections.

4.2 Storage Silo Implementation

While my objective is to construct *Indaleko*, which will support multiple silos combined with federated naming, activity, and meta-data services, my proposed initial step is to do this on a single system consisting of two storage silos.

This section constitutes an initial "straw person" proposal for what I

anticipate constructing. In doing so I expect to evaluate at least two of the following research questions:

Can the separation of naming from the file system(s) be achieved without compromising
This is related to the broader question articulated earlier in this chapter (Section 4.1.)

Can this two-silo naming system be used to demonstrate more effective data location?
The original motivation is that rich, dynamic naming will enable "finding my stuff". One approach to this would be to build on top of prior work done in our department related to *data curation* to see if a richer naming system can be used to simplify that task [50].

Can a non-traditional storage mechanism be effective when used as a storage silo within
This specifically speaks to the non-hierarchical name system idea that was previously proposed in "Hierarchical File Systems Are Dead" [7]. While not a primary motivator for the main thesis, I am intrigued by the idea of taking a persistent memory key-value store along the lines of "Modernizing File System through In-Storage Indexing" [6] where the key-value store is in persistent memory, which allows me to leverage my own prior work.

Does the explicit separation of meta-data and naming from the file system enable the cr
Traditional namespaces are static in that the names of objects within them are not changed by the system, yet relationship based namespaces could be dynamic. Further, the namespace of interest to me now may not be the same as the namespace of interest to someone using the same data but in the context of a different role — by not tying the namespace to the data does it provide a more functional namespace?
TM ▶*This question is not fleshed out enough at this point.*◀

Does capturing activity context related to the way that data is constructed and used all
One obvious way of demonstrating this might be to show where it does permit that type of enhanced data management mechanism. For example, this might be related to the data curation aspect of prior work. It is likely there are other similar data management activities that might yield a useful basis for finding information of interest.

I anticipate that this work will require developing specific components, which largely correspond to the services described in Section 3.3.1.

One key aspect that has not been well-defined in the model section but is important to do as part of this work is to define what an *activity*

context is, how it is created, and how it is used within the system. This will be an important aspect of this work, even if the actual implementation is quite simple, possibly capturing only a tiny amount of information. Initial techniques for doing this can likely be based upon existing information gathering systems, such as *eBPF* TM ►*add to glossary, define out*◄ in Linux or *ETW* TM ►*add to glossary, define out*◄ on Windows, both of which are pre-existing systems with well-defined mechanisms for extracting information from production systems.

To review, the services that will be implemented in this phase of my work are:

Metadata Servers — there are existing models for meta-data servers, such as Egeria [1] or Amundsen [2] (a demonstrative, but not definitive list) that might be sufficient. If not, using a key-value store such as WiredTiger [3] to construct a metadata server is also viable. TM ►*I'd rather not build anything I don't have to but my concern is that this is going to be a core bit of the work; the right thing to do is to do a more thorough search for these technologies and figure out which one looks like a good fit.* Building *one is possible but seems like a big project on its own.*◄

Namespace Servers — my expectation is that this will need to be constructed; it must interoperate with the metadata servers. Work here will include defining the interface to constructing a new namespace or retrieving a previously constructed namespace. While the ultimate goal is to have a federated namespace service, this initial effort need only provide a single, non-federated namespace even though that is more limited than the actual design.

Activity Monitors — these create the activity context information I described previously. I would expect that only a single activity monitor will be built for this initial system prototype and the details of this will be driven by the chosen definition of the activity context and relevant activity providers.

Attribute Services — because *Indaleko* is a naming system, we need a mechanism that retrieves storage attributes. Initially, this will consist of a source for scanning current content and then a monitoring mechanism

[1] https://github.com/odpi/egeria
[2] https://github.com/amundsen-io/amundsen
[3] https://github.com/wiredtiger/wiredtiger

(likely similar to the activity provider) that notifies the attribute service to update the attributes of files that have changed.

Update Notification Server — existing systems provide change notifications. In a cross-silo system this sort of dynamic state monitoring will also be useful. An initial implementation for a local system could consist simply of using the existing change notification mechanism(s) for watching changes. A more robust implementation could use information from the meta-data service(s) to determine if a given change is of interest. This dynamic change tracking could then be federated as part of the later planned work.

This proposal is quite broad and consists of no working software at the present time. To achieve this I propose taking the initial architecture and constructing a design based upon this architecture. To evaluate the design, I propose choosing at least two distinct storage silos and at least one target operating system. To the extent possible, initial implementation would focus on combining existing components as much as possible. For example, Using MinIo and Sparkle Share as storage silos, with Windows Cloud Sync would provide documented and generally well-understood technologies on which to construct these services. The Meta-data server can be constructed using one of the available key-value stores (e.g., WiredTiger). The Notification service could start with Emitter. Initial activity context work can be built using eBPF or other inbuilt tracing mechanisms. This leaves how to build the attribute services as an open area to be further refined. TM ▶*TBD*◀

4.3 Federated Naming/Meta-data Security

TM ▶*TBD*◀

4.4 Cross-Silo Naming/Meta-data Benefits

TM ▶*TBD*◀

Chapter 5

Timeline

TM ►*This section is TBD*◄

Bibliography

[1] J. Locke, *Locke's essays: An essay concerning human understanding, and A treatise on the conduct of the understanding (Complete in 1 volume with the author's last additions and corrections).* 1844 (**backrefpage** 1).

[2] V. Bush **andothers**, "As we may think," *The atlantic monthly*, **journvol** 176, **number** 1, **pages** 101–108, 1945 (**backrefpages** 1, 17, 18, 73).

[3] P. J. Guo **and** M. Seltzer, "Burrito: Wrapping your lab notebook in computational infrastructure," **in***TaPP'12 Proceedings of the 4th USENIX conference on Theory and Practice of Provenance* 2012, **pages** 7–7 (**backrefpages** 4, 11, 16, 79, 84, 88, 89).

[4] M. K. McKusick, W. N. Joy, S. J. Leffler **and** R. S. Fabry, "A fast file system for unix," *ACM Transactions on Computer Systems*, **journvol** 2, **number** 3, **pages** 181–197, 1984 (**backrefpage** 5).

[5] H. Custer, *Inside the Windows NT File System*. 1994 (**backrefpage** 5).

[6] J. Koo, J. Im, J. Song, J. Park, E. Lee, B. S. Kim **and** S. Lee, "Modernizing file system through in-storage indexing," **in***15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)* 2021, **pages** 75–92 (**backrefpages** 5, 27).

[7] M. Seltzer **and** N. Murphy, "Hierarchical file systems are dead," **in***HotOS'09 Proceedings of the 12th conference on Hot topics in operating systems* 2009, **pages** 1–1 (**backrefpages** 5, 27).

[8] P. Braam, "The lustre storage architecture," *arXiv preprint arXiv:1903.01955*, 2019 (**backrefpage** 5).

[9] R. Noronha **and** D. Panda, "Imca: A high performance caching front-end for glusterfs on infiniband," **in***2008 37th International Conference on Parallel Processing* 2008, **pages** 462–469 (**backrefpage** 5).

[10] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long **and** C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," **in***Proceedings of the 7th symposium on Operating systems design and implementation* 2006, **pages** 307–320 (**backrefpage** 5).

[11] Y. Kawai, A. Hasan, G. Iwai, T. Sasaki **and** Y. Watase, "A method for reliably managing files with rns in multi data grids," **in***Procedia Computer Science* **volume** 4, 2011, **pages** 412–421 (**backrefpage** 5).

[12] K. Briney, *Data Management for Researchers: Organize, maintain and share your data for research success*. 2015 (**backrefpage** 6).

[13] T. Berners-Lee, R. Fielding **and** L. Masinter, "Uniform resource identifiers (uri): Generic syntax," *RFC*, **jourvol** 2396, **pages** 1–40, 1998 (**backrefpage** 7).

[14] S. R. Mashwani **and** S. Khusro, "360° semantic file system: Augmented directory navigation for nonhierarchical retrieval of files," *IEEE Access*, **jourvol** 7, **pages** 9406–9418, 2019 (**backrefpage** 11).

[15] M. .-. Vef, R. Steiner, R. Salkhordeh, J. Steinkamp, F. Vennetier, J. .-. Smigielski **and** A. Brinkmann, "Delvefs - an event-driven semantic file system for object stores," **in***2020 IEEE International Conference on Cluster Computing (CLUSTER)* 2020, **pages** 35–46. DOI: 10.1109/CLUSTER49012.2020.00014 (**backrefpages** 11, 79, 85).

[16] M.-A. Vef, R. Steiner, R. Salkhordeh, J. Steinkamp, F. Vennetier, J.-F. Smigielski **and** A. Brinkmann, "Delvefs - an event-driven semantic file system for object stores," **in***2020 IEEE International Conference on Cluster Computing (CLUSTER)* 2020, **pages** 35–46 (**backrefpage** 11).

[17] P. Dourish, "The appropriation of interactive technologies: Some lessons from placeless documents," *conference on computer supported cooperative work*, **jourvol** 12, **number** 4, **pages** 465–490, 2003 (**backrefpage** 11).

[18] S. Harrison **and** P. Dourish, "Re-place-ing space: The roles of place and space in collaborative systems," **in***Proceedings of the 1996 ACM conference on Computer supported cooperative work* 1996, **pages** 67–76 (**backrefpage** 11).

[19] D. Barreau **and** B. A. Nardi, "Finding and reminding: File organization from the desktop," *ACM Sigchi Bulletin*, **jourvol** 27, **number** 3, **pages** 39–43, 1995 (**backrefpage** 11).

[20] P. Dourish, R. Bentley, R. Jones **and** A. MacLean, "Getting some perspective: Using process descriptions to index document history," **in***Proceedings of the international ACM SIGGROUP conference on Supporting group work* 1999, **pages** 375–384 (**backrefpage** 11).

[21] P. Dourish, W. K. Edwards, A. LaMarca, J. Lamping, K. Petersen, M. Salisbury, D. B. Terry **and** J. Thornton, "Extending document management systems with user-specific active properties," *ACM Trans. Inf. Syst.*, **jourvol** 18, **number** 2, **pages** 140–170, **april** 2000, ISSN: 1046-8188. DOI: 10.1145/348751.348758. **url**: https://doi.org/10.1145/348751.348758 (**backrefpages** 11, 87).

[22] D. K. Gifford, P. Jouvelot, M. A. Sheldon **and** J. W. O'Toole, "Semantic file systems," **in***Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles* **jourser** SOSP '91, Pacific Grove, California, USA: Association for Computing Machinery, 1991, **pages** 16–25, ISBN: 0897914473. DOI: 10.1145/121132.121138. **url**: https://doi.org/10.1145/121132.121138 (**backrefpages** 11, 96).

[23] R. Pike, D. Presotto, K. Thompson, H. Trickey **and** P. Winterbottom, "The use of name spaces in plan 9," *SIGOPS Oper. Syst. Rev.*, **jourvol** 27, **number** 2, **pages** 72–76, **april** 1993, ISSN: 0163-5980. DOI: 10.1145/155848.155861. **url**: https://doi.org/10.1145/155848.155861 (**backrefpages** 11, 96).

[24] M. A. Olson, "The design and implementation of the inversion file system," **in***USENIX Winter 1993 Conference (USENIX Winter 1993 Conference)* San Diego, CA: USENIX Association, **january** 1993. **url**: https://www.usenix.org/conference/usenix-winter-1993-conference/design-and-implementation-inversion-file-system (**backrefpages** 11, 96).

[25] Y. Hua, H. Jiang, Y. Zhu, D. Feng **and** L. Tian, "Smartstore: A new metadata organization paradigm with semantic-awareness for next-generation file systems," **in***Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* 2009, **pages** 1–12. DOI: 10.1145/1654059.1654070 (**backrefpages** 11, 21, 92, 96).

[26] S. Bloehdorn, O. Görlitz, S. Schenk **and** M. Völkel, "Tagfs - tag semantics for hierarchical file systems," **in***Proceedings of the 6th International Conference on Knowledge Management (I-KNOW 06), Graz, Austria, September 6-8, 2006* **september** 2006 (**backrefpages** 11, 96).

[27] D. Di Sarli **and** F. Geraci, "Gfs: A graph-based file system enhanced with semantic features," **in***Proceedings of the 2017 International Conference on Information System and Data Mining* **jourser** ICISDM '17, Charleston, SC, USA: Association for Computing Machinery, 2017, **pages** 51–55, ISBN: 9781450348331. DOI: 10.1145/3077584.3077591. **url**: https://doi.org/10.1145/3077584.3077591 (**backrefpages** 11, 96).

[28] S. Shah, C. A. N. Soules, G. R. Ganger **and** B. D. Noble, "Using provenance to aid in personal file search," **in***2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference* **jourser** ATC'07, Santa Clara, CA: USENIX Association, 2007, ISBN: 9998888776 (**backrefpages** 11, 96).

[29] A. A. Rasel **and** M. E. Ali, "Uprove2: Privacy-aware, scalable, ubiquitous provenance to enhance file search," **in***2016 International Conference on Networking Systems and Security (NSysS)* IEEE, 2016, **pages** 1–5 (**backrefpages** 11, 96).

[30] J. Liu, D. Feng, Y. Hua, B. Peng, P. Zuo **and** Y. Sun, "P-index: An efficient searchable metadata indexing scheme based on data provenance in cold storage," **in***International Conference on Algorithms and Architectures for Parallel Processing* Springer, 2015, **pages** 597–611 (**backrefpages** 11, 96).

[31] L. Page, S. Brin, R. Motwani **and** T. Winograd, "The pagerank citation ranking: Bringing order to the web.," Stanford InfoLab, techreport, 1999 (**backrefpages** 11, 74, 96).

[32] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh **and** B. Lyon, "Design and implementation of the sun network filesystem," **in***Proceedings of the Summer USENIX conference* 1985, **pages** 119–130 (**backrefpages** 11, 96).

[33] D. Bermbach, M. Klems, S. Tai **and** M. Menzel, "Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs," **in***2011 IEEE 4th International Conference on Cloud Computing* 2011, **pages** 452–459. DOI: 10.1109/CLOUD.2011.62 (**backrefpages** 11, 97).

[34] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham **and** M. J. West, "Scale and performance in a distributed file system," *ACM Transactions on Computer Systems*, **jourvol** 6, **number** 1, **pages** 51–81, 1988 (**backrefpages** 11, 78, 96).

[35] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal **and** F. D. Smith, "Andrew: A distributed personal computing environment," *Commun. ACM*, **jourvol** 29, **number** 3, **pages** 184–201, **march** 1986, ISSN: 0001-0782. DOI: 10.1145/5666.5671. **url**: https://doi.org/10.1145/5666.5671 (**backrefpage** 11).

[36] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer **and** R. P. Wattenhofer, "Farsite: Federated, available, and reliable storage for an incompletely trusted environment," *SIGOPS Oper. Syst. Rev.*, **jourvol** 36, **number** SI, **pages** 1–14, **december** 2003, ISSN: 0163-5980. DOI: 10.1145/844128.844130. **url**: https://doi.org/10.1145/844128.844130 (**backrefpages** 11, 97).

[37] D. M. Ritchie **and** K. Thompson, "The unix time-sharing system," *Commun. ACM*, **jourvol** 17, **number** 7, **pages** 365–375, **july** 1974, ISSN: 0001-0782. DOI: 10.1145/361011.361061. **url**: https://doi.org/10.1145/361011.361061 (**backrefpages** 11, 96).

[38] A. N. Bessani, R. Mendes, T. Oliveira, N. F. Neves, M. Correia, M. Pasin **and** P. Verissimo, "Scfs: A shared cloud-backed file system.," **in***USENIX Annual Technical Conference* Citeseer, 2014, **pages** 169–180 (**backrefpages** 11, 97).

[39] C. Wang, K. Thareja, M. Stealey, P. Ruth **and** I. Baldin, "Comet: A distributed metadata service for federated cloud infrastructures," **in***2019 IEEE High Performance Extreme Computing Conference (HPEC)* 2019, **pages** 1–7. DOI: 10.1109/HPEC.2019.8916536 (**backrefpages** 11, 21, 92, 97).

[40] Y. Demchenko, C. Ngo, C. de Laat **and** C. Lee, "Federated access control in heterogeneous intercloud environment: Basic models and architecture patterns," **in***2014 IEEE International Conference on Cloud Engineering* 2014, **pages** 439–445. DOI: 10.1109/IC2E.2014.84 (**backrefpage** 11).

[41] J. Benet, "Ipfs - content addressed, versioned, p2p file system.," *arXiv preprint arXiv:1407.3561*, 2014 (**backrefpages** 11, 97).

[42] M. L. Mazurek, Y. Liang, W. Melicher, M. Sleeper, L. Bauer, G. R. Ganger, N. Gupta **and** M. K. Reiter, "Toward strong, usable access control for shared distributed data," **in***Proceedings of the 12th USENIX conference on File and Storage Technologies* 2014, **pages** 89–103 (**backrefpage** 11).

[43] Y. Li, N. S. Dhotre, Y. Ohara, T. M. Kroeger, E. L. Miller **and** D. D. E. Long, "Horus: Fine-grained encryption-based security for large-scale storage," **in***Proceedings of the 11th USENIX conference on File and Storage Technologies* 2013, **pages** 147–160 (**backrefpage** 11).

[44] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer **and** R. P. Wattenhofer, "Farsite: Federated, available, and reliable storage for an incompletely trusted environment," **in***Proceedings of the 5th symposium on Operating systems design and implementation (Copyright restrictions prevent ACM from being able to make the PDFs for this conference available for downloading)* **volume** 36, 2002, **pages** 1–14 (**backrefpage** 11).

[45] L. Carata, S. Akoush, N. Balakrishnan, T. Bytheway, R. Sohan, M. Seltzer **and** A. Hopper, "A primer on provenance," *Commun. ACM*, **jourvol** 57, **number** 5, **pages** 52–60, **may** 2014, ISSN: 0001-0782. DOI: 10.1145/2596628. **url**: https://doi.org/10.1145/2596628 (**backrefpages** 11, 95).

[46] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer **and** J. Bacon, "Practical whole-system provenance capture," **in***Symposium on Cloud Computing (SoCC'17)* ACM, 2017 (**backrefpages** 11, 23, 94).

[47] K. J. Vicente **and** R. C. Williges, "Accommodating individual differences in searching a hierarchical file system," *International Journal of Human-computer Studies International Journal of Man-machine Studies*, **jourvol** 29, **number** 6, **pages** 647–668, 1988 (**backrefpage** 12).

[48] J. Parkinson **and** Q. Cutts, "Investigating the relationship between spatial skills and computer science," **august** 2018, **pages** 106–114. DOI: 10.1145/3230977.3230990 (**backrefpage** 12).

[49] O. Bergman, T. Israeli **and** S. Whittaker, "Search is the future? the young search less for files," **in***Proceedings of the Association for Information Science and Technology* **volume** 56, 2019, **pages** 360–363 (**backrefpage** 13).

[50] F. Vitale, "Personal data curation in the cloud age : Individual differences and design opportunities," phdthesis, University of British Columbia, 2020. DOI: http://dx.doi.org/10.14288/1.0392427. **url**: https://open.library.ubc.ca/collections/ubctheses/24/items/1.0392427 (**backrefpages** 13, 27).

[51] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, M. Marchukov, D. Petrov, L. Puzar, Y. J. Song **and** V. Venkataramani, "TAO: Facebook's distributed data store for the social graph," **in***Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)* San Jose, CA: USENIX, 2013, **pages** 49–60, ISBN: 978-1-931971-01-0. **url**: https://www.usenix.org/conference/atc13/technical-sessions/presentation/bronson (**backrefpage** 69).

[52] M. A. Olson **andothers**, "The design and implementation of the inversion file system," **in***USENIX Winter* 1993, **pages** 205–218 (**backrefpage** 69).

[53] P. J. Shetty, R. P. Spillane, R. R. Malpani, B. Andrews, J. Seyster **and** E. Zadok, "Building workload-independent storage with vt-trees," **in***Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 13)* San Jose, CA: USENIX,

2013, **pages** 17–30, ISBN: 978-1-931971-99-7. **url**: https://www.usenix.org/conference/fast13/technical-sessions/presentation/shetty (**backrefpage** 69).

[54] J. Masci, M. M. Bronstein, A. M. Bronstein **and** J. Schmidhuber, "Multimodal similarity-preserving hashing," *IEEE transactions on pattern analysis and machine intelligence*, **jourvol** 36, **number** 4, **pages** 824–830, 2014 (**backrefpage** 70).

[55] R. Pike, D. Presotto, K. Thompson, H. Trickey **and** P. Winterbottom, "The use of name spaces in plan 9," **in***Proceedings of the 5th workshop on ACM SIGOPS European workshop: Models and paradigms for distributed systems structuring* ACM, 1992, **pages** 1–5 (**backrefpage** 70).

[56] C. A. Soules **and** G. R. Ganger, "Toward automatic context-based attribute assignment for semantic file systems," *Parallel data laboratory, Carnegie Mellon University. June*, 2004 (**backrefpages** 71, 74).

[57] S. Bloehdorn, O. Görlitz, S. Schenk, M. Völkel **andothers**, "Tagfs-tag semantics for hierarchical file systems," **in***Proceedings of the 6th International Conference on Knowledge Management (I-KNOW 06), Graz, Austria* **volume** 8, 2006, **pages** 6–8 (**backrefpage** 71).

[58] C. A. Soules **and** G. R. Ganger, "Why can't i find my files?: New methods for automating attribute assignment," **in***HotOS* 2003, **pages** 115–120 (**backrefpages** 71, 74).

[59] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer **and** J. Bacon, "Practical whole-system provenance capture," **in***Proceedings of the 2017 Symposium on Cloud Computing* **jourser** SoCC '17, Santa Clara, California: ACM, 2017, **pages** 405–418, ISBN: 978-1-4503-5028-0. DOI: 10.1145/3127479.3129249. **url**: http://doi.acm.org/10.1145/3127479.3129249 (**backrefpages** 71, 75).

[60] M. Factor, K. Meth, D. Naor, O. Rodeh **and** J. Satran, "Object storage: The future building block for storage systems," **in***Local to Global Data Interoperability-Challenges and Technologies, 2005* IEEE, 2005, **pages** 119–123 (**backrefpage** 72).

[61] Y. Mao, E. Kohler **and** R. T. Morris, "Cache craftiness for fast multicore key-value storage," **in***Proceedings of the 7th ACM european conference on Computer Systems* ACM, 2012, **pages** 183–196 (**backrefpage** 72).

[62] M. Rudolf, M. Paradies, C. Bornhövd **and** W. Lehner, "The graph story of the sap hana database.," **in***BTW* Citeseer, **volume** 13, 2013, **pages** 403–420 (**backrefpage** 72).

[63] J. Webber **and** I. Robinson, *A programmatic introduction to neo4j*. Addison-Wesley Professional, 2018 (**backrefpage** 72).

[64] Microsoft, *Welcome to azure cosmos db*, (accessed January 17, 2019), **january** 2019. **url**: https://docs.microsoft.com/en-us/azure/cosmos-db/introduction (**backrefpage** 72).

[65] A. Kyrola, G. E. Blelloch **and** C. Guestrin, "Graphchi: Large-scale graph computation on just a pc," USENIX, 2012 (**backrefpage** 72).

[66] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin **and** J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041*, 2014 (**backrefpage** 72).

[67] D. Nguyen, A. Lenharth **and** K. Pingali, "A lightweight infrastructure for graph analytics," **in***Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* ACM, 2013, **pages** 456–471 (**backrefpage** 72).

[68] S. Salihoglu **and** J. Widom, "Gps: A graph processing system," **in***Proceedings of the 25th International Conference on Scientific and Statistical Database Management* ACM, 2013, **page** 22 (**backrefpage** 72).

[69] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser **and** G. Czajkowski, "Pregel: A system for large-scale graph processing," **in***Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* ACM, 2010, **pages** 135–146 (**backrefpage** 72).

[70] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin **and** I. Stoica, "Graphx: Graph processing in a distributed dataflow framework.," **in***OSDI* **volume** 14, 2014, **pages** 599–613 (**backrefpage** 72).

[71] J. Shun **and** G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," **in***ACM Sigplan Notices* ACM, **volume** 48, 2013, **pages** 135–146 (**backrefpage** 72).

[72] P. Macko, V. J. Marathe, D. W. Margo **and** M. I. Seltzer, "Llama: Efficient graph analytics using large multiversioned arrays," **in***Proceedings of the 31st IEEE International Conference on Data Engineering* IEEE, 2015 (**backrefpage** 72).

[73] D. Margo **and** M. Seltzer, "A scalable distributed graph partitioner," *Proceedings of the VLDB Endowment*, **jourvol** 8, **number** 12, **pages** 1478–1489, 2015 (**backrefpage** 73).

[74] R. Sandberg, "The sun network file system: Design, implementation and experience," **in***in Proceedings of the Summer 1986 USENIX Technical Conference and Exhibition* Citeseer, 1986 (**backrefpage** 73).

[75] B. Sidebotham **andothers**, *Volumes: the andrew file system data structuring primitive*. Carnegie Mellon University, Information Technology Center, 1986 (**backrefpage** 73).

[76] M. K. Sreenivas, S. R. Steiner, A. Brjazovski **and** S. H. Agarwal, *Bypass of the namespace hierarchy to open files*, US Patent 7,925,681, **april** 2011 (**backrefpage** 73).

[77] G. Watumull, J. Gerend, L. Poggemeyer, A. Hansen **and** K. Ainapure, *Resilient file system (refs) overview*, (accessed January 17, 2019). **url**: https://tinyurl.com/y832s3ky (**backrefpage** 73).

[78] S. Ghemawat, H. Gobioff **and** S.-t. Leung, "Google_File_System," 2003, ISSN: 01635980. DOI: 10.1145/1165389.945450. arXiv: z0024 (**backrefpages** 73, 75).

[79] S. Dumais, E. Cutrell, J. J. Cadiz, G. Jancke, R. Sarin **and** D. C. Robbins, "Stuff i've seen: A system for personal information retrieval and re-use," **in***ACM SIGIR Forum* ACM, **volume** 49, 2016, **pages** 28–35 (**backrefpage** 73).

[80] M. Arenas, B. C. Grau, E. Kharlamov, Š. Marciuška **and** D. Zheleznyakov, "Faceted search over rdf-based knowledge graphs," *Web Semantics: Science, Services and Agents on the World Wide Web*, **jourvol** 37, **pages** 55–74, 2016 (**backrefpage** 73).

[81] D. Tunkelang, "Faceted search," *Synthesis lectures on information concepts, retrieval, and services*, **jourvol** 1, **number** 1, **pages** 1–80, 2009 (**backrefpage** 73).

[82] M. Hearst, "Design recommendations for hierarchical faceted search interfaces," **in***ACM SIGIR workshop on faceted search* Seattle, WA, 2006, **pages** 1–5 (**backrefpage** 73).

[83] V. Klungre **and** M. Giese, "Evaluating a faceted search index for graph data," **in***OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* Springer, 2018, **pages** 573–583 (**backrefpage** 73).

[84] R. Walton, "Looking for answers: A usability study of online finding aid navigation," *The American Archivist*, **jourvol** 80, **number** 1, **pages** 30–52, 2017 (**backrefpage** 73).

[85] P. H. Cleverley **and** S. Burnett, "Retrieving haystacks: A data driven information needs model for faceted search," *Journal of Information Science*, **jourvol** 41, **number** 1, **pages** 97–113, 2015 (**backrefpage** 73).

[86] Apple, *Search and spotlight*, (accessed January 6, 2019). **url**: https://tinyurl.com/yasf7kf2 (**backrefpage** 74).

[87] Microsoft, *Data add-in interfaces*, (accessed January 6, 2019). **url**: https://tinyurl.com/y83elc34 (**backrefpage** 74).

[88] J. Chou, "Findfs: Adding tag-based views to a hierarchical filesystem," phdthesis, University of British Columbia, 2015 (**backrefpage** 74).

[89] S. Ma **and** S. Wiedenbeck, "File management with hierarchical folders and tags," **in***CHI'09 Extended Abstracts on Human Factors in Computing Systems* ACM, 2009, **pages** 3745–3750 (**backrefpage** 74).

[90] A. Laursen, "A novel, tag-based file-system," (accessed March 29, 2018), mathesis, Macalester College, 2014. **url**: https://digitalcommons.macalester.edu/mathcs%5C_honors/34/ (**backrefpage** 74).

[91] nayuki, *Designing better file organization around tags, not hierarchies*, (accessed October 1, 2018), **march** 2017. **url**: https://www.nayuki.io/page/designing-better-file-organization-around-tags-not-hierarchies (**backrefpages** 74, 76).

[92] P. Andrews, I. Zaihrayeu **and** J. Pane, "A classification of semantic annotation systems," *Semantic Web*, **jourvol** 3, **number** 3, **pages** 223–248, 2012, ISSN: 15700844. DOI: 10.3233/SW-2011-0056 (**backrefpages** 74, 76).

[93] L. Up, "Tag , no di !," **pages** 1–23, 2016 (**backrefpage** 74).

[94] I. Jones, W. You **and** C. A. N. Do, "Tagxfs is a semantic file system. It extends the user space file system to a tag based hierarchy.," **pages** 1–5, 2016 (**backrefpage** 74).

[95] A. Parker-Wood, D. D. E. Long, E. Miller, P. Rigaux **and** A. Isaacson, "A File By Any Other Name: Managing File Names with Metadata," *Proceedings of International Conference on Systems and Storage*, **pages** 1–11, 2014 (**backrefpage** 74).

[96] I. Amazon Web Services, *Object tagging*, (accessed January 8, 2019), 2018. **url**: https://docs.aws.amazon.com/AmazonS3/latest/dev/object-tagging.html (**backrefpage** 74).

[97] S. Ames, N. Bobb, K. M. Greenan, O. S. Hofmann, M. W. Storer, C. Maltzahn, E. L. Miller **and** S. A. Brandt, "Lifs: An attribute-rich file system for storage class memories," **in***Proceedings of the 23rd IEEE/14th NASA Goddard Conference on Mass Storage Systems and Technologies* IEEE, 2006 (**backrefpage** 74).

[98] A. Leung, I. Adams **and** E. L. Miller, "Magellan: A searchable metadata architecture for large-scale file systems," *University of California, Santa Cruz, Tech. Rep. UCSC-SSRC-09-07*, 2009 (**backrefpage** 74).

[99] O. Frieder **and** S. Kapoor, *Hierarchical structured abstract data organization system*, US Patent 8,209,358, **june** 2012 (**backrefpage** 74).

[100] Y. Hua, H. Jiang **and** D. Feng, "Real-time semantic search using approximate methodology for large-scale storage systems," *IEEE Transactions on Parallel and Distributed Systems*, **jourvol** 27, **number** 4, **pages** 1212–1225, 2016 (**backrefpage** 74).

[101] D. Di Sarli **and** F. Geraci, "Gfs: A graph-based file system enhanced with semantic features," **in***Proceedings of the 2017 International Conference on Information System and Data Mining* ACM, 2017, **pages** 51–55 (**backrefpage** 74).

[102] B. Martin, "Formal concept analysis and semantic file systems," **in***International Conference on Formal Concept Analysis* Springer, 2004, **pages** 88–95 (**backrefpage** 74).

[103] ——, "Formal concept analysis and semantic file systems," phdthesis, University of Wollongong, 2008. **url**: https://ro.uow.edu.au/theses/260/ (**backrefpage** 74).

[104] ——, "Open source libferris: Chasing the "everything is a file system" dream," *linux.com*, **january** 2014, (accessed September 4, 2018). **url**: https://www.linux.com/learn/open-source-libferris-chasing-everything-file-system-dream (**backrefpages** 74, 76).

[105] S. Faubel **and** C. Kuschel, "Towards semantic file system interfaces," *CEUR Workshop Proceedings*, **journvol** 401, 2008, ISSN: 16130073 (**backrefpage** 74).

[106] M. Harlan **and** G. Parmer, "Joinfs: A semantic file system for embedded systems," **in***Proceedings of the International Conference on Embedded Systems and Applications (ESA)* The Steering Committee of The World Congress in Computer Science, Computer Engineering **and** Applied Computing (WorldComp), 2011, **page** 1 (**backrefpage** 74).

[107] M. Suguna **and** T. Anand, "Dynamic Metadata Management in Semantic File Systems," **journvol** 5, **number** 3, **pages** 44–47, 2015 (**backrefpage** 74).

[108] B.-H. Ngo, C. Bac **and** F. Silber-Chaussumier, "Integrating ontology into semantic file systems," **in***Huitièmes Journées Doctorales en Informatique et Réseaux (JDIR'07)* 2007, **pages** 139–142 (**backrefpage** 74).

[109] P. Omvlee, "A novel idea for a new Filesystem," *June*, 2009. **url**: http://referaat.cs.utwente.nl/conference/11/paper/6966/a-novel-idea-for-a-new-filesystem.pdf (**backrefpages** 74, 76).

[110] G. Wang, H. Lu, G. Yu **and** B. YuBao, "Managing very large document collections using semantics," *Journal of Computer Science and Technology*, **journvol** 18, **number** 3, **pages** 403–406, 2003 (**backrefpage** 74).

[111] B. Gopal **and** U. Manber, "Integrating content-based access mechanisms with hierarchical file systems," **in***OSDI* **volume** 99, 1999, **pages** 265–278 (**backrefpage** 74).

[112] B. Martin **and** P. Eklund, "Applying formal concept analysis to semantic file systems leveraging wordnet," *ADCS 2005 - Proceedings of the Tenth Australasian Document Computing Symposium*, **pages** 56–63, 2005 (**backrefpage** 74).

[113] V. Codocedo **and** A. Napoli, "Formal Concept Analysis and Information Retrieval – A Survey," *Formal Concept Analysis: 13th International Conference, ICFCA 2015*, **pages** 61–77, 2015. DOI: 10.1007/978-3-319-19545-2_4. **url**: http://dx.doi.org/10.1007/978-3-319-19545-2%7B%5C_%7D4 (**backrefpage** 74).

[114] M. Mahalingam, C. Tang **and** Z. Xu, "Towards a semantic, deep archival file system," *Proceedings of the IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, **jourvol** 2003-Janua, **pages** 115–121, 2003. DOI: 10.1109/FTDCS.2003.1204321 (**backrefpage** 74).

[115] D. K. Gifford, P. Jouvelot, M. A. Sheldon **and** J. W. O'Toole, "Semantic file systems," **in***Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles* **jourser** SOSP '91, Pacific Grove, California, USA: Association for Computing Machinery, 1991, **pages** 16–25, ISBN: 0897914473. DOI: 10.1145/121132.121138. **url**: https://doi.org/10.1145/121132.121138 (**backrefpages** 74, 78).

[116] M. Seltzer **and** N. Murphy, "Hierarchical file systems are dead," **in***Proceedings of the 12th Conference on Hot Topics in Operating Systems* **jourser** HotOS'09, Monte Verità, Switzerland: USENIX Association, 2009, **page** 1 (**backrefpages** 74, 78).

[117] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun **and** M. Seltzer, "Provenance-aware storage systems," **in***Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference* **jourser** ATEC '06, Boston, MA: USENIX Association, 2006, **pages** 4–4. **url**: http://dl.acm.org/citation.cfm?id=1267359.1267363 (**backrefpage** 75).

[118] R. Harper, S. Lindley, E. Thereska, R. Banks, P. Gosset, G. Smyth, W. Odom **and** E. Whitworth, "What is a file?" **in***Proceedings of the 2013 conference on Computer supported cooperative work* ACM, 2013, **pages** 1125–1136 (**backrefpage** 76).

[119] S. E. Lindley, G. Smyth, R. Corish, A. Loukianov, M. Golembewski, E. A. Luger **and** A. Sellen, "Exploring new metaphors for a networked world through the file biography," **in***Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* ACM, 2018, **page** 118 (**backrefpage** 76).

[120] M. T. Khan, M. Hyun, C. Kanich **and** B. Ur, "Forgotten but not gone: Identifying the need for longitudinal data management in cloud storage," **in***Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* ACM, 2018, **page** 543 (**backrefpage** 76).

[121] F. Vitale, I. Janzen **and** J. McGrenere, "Hoarding and minimalism: Tendencies in digital data preservation," **in***Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* ACM, 2018, **page** 587 (**backrefpage** 76).

[122] R. Boardman, R. Spence **and** M. A. Sasse, "Too many hierarchies? the daily struggle for control of the workspace," **in***Proceedings of HCI international* **volume** 1, 2003, **pages** 616–620 (**backrefpage** 76).

[123] S. Jan, M. Li, G. Al-Sultany, H. Al-Raweshidy **and** I. A. Shah, "Semantic file annotation and retrieval on mobile devices," *Mobile Information Systems*, **jourvol** 7, **number** 2, **pages** 107–122, 2011, ISSN: 1574017X. DOI: 10.3233/MIS-2011-0113 (**backrefpage** 76).

[124] R. Mander, G. Salomon **and** Y. Y. Wong, "A "Pile" Metaphor for Supporting Casual Organization of Information," *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92*, **pages** 627–634, 1992. DOI: 10.1145/142750.143055. **url**: http://portal.acm.org/citation.cfm?doid=142750.143055 (**backrefpage** 76).

[125] Apple, Inc., *Search drives user engagement*, 2016 (**backrefpage** 77).

[126] O. Bergman, T. Israeli **and** S. Whittaker, "Factors hindering shared files retrieval," *Aslib Journal of Information Management*, **jourvol** 72, **number** 1, **pages** 130–147, 2019 (**backrefpages** 77, 78).

[127] A. authors, *Anonymized title* (**backrefpage** 78).

[128] M. L. Kazar, B. W. Leverett, O. T. Anderson, V. Apostolides, B. A. Bottos, S. Chutani, C. Everhart, W. A. Mason, S.-T. Tu **and** E. R. Zayas, "Decorum file system architectural overview.," *USENIX Summer*, **pages** 151–164, 1990 (**backrefpage** 78).

[129] R. Pike **and** P. Weinberger, "The Hideous Name," **in***USENIX Summer 1985 Conference Proceedings* Portland Oregon, 1985 (**backrefpage** 78).

[130] J. C. Mogul, "Representing information about files," phdthesis, Stanford University, 1986 (**backrefpage** 78).

[131] R. Pike, D. Presotto, K. Thompson, H. Trickey **and** P. Winterbottom, "The use of name spaces in plan 9," *SIGOPS Oper. Syst. Rev.*, **journvol** 27, **number** 2, **pages** 72–76, **april** 1993, ISSN: 0163-5980. DOI: 10.1145/155848.155861. **url**: https://doi.org/10.1145/155848.155861 (**backrefpage** 78).

[132] A. R. Fox, A. Satyarnarayan, P. Guo, H. Xia **and** J. D. Hollan, *Chs: Medium: A human-centered information space:designing dynamic personalized visual information*, 2019. **url**: http://hci.ucsd.edu/220/NSF_Hollan_220.pdf (**backrefpage** 78).

[133] F. Vitale, "Personal data curation in the cloud age: Individual differences and design opportunities," phdthesis, University of British Columbia, 2020 (**backrefpage** 78).

[134] T. W. Malone, "How do people organize their desks?: Implications for the design of office information systems," *ACM Transactions on Information Systems*, **journvol** 1, **number** 1, **pages** 99–112, 1983 (**backrefpage** 78).

[135] Google Inc., *Google Cloud Storage API - Blobs/Objects*, Version 1.35.0. Online. Accessed 2021-02-01. https://googleapis.dev/python/storage/latest/blobs.html, 2019 (**backrefpage** 79).

[136] Amazon Web Services, Inc, *Amazon Simple Storage Service Developer Guide*, API Version 2006-03-01. Online. Accessed 2021-02-01. https://docs.aws.amazon.com/AmazonS3/latest/dev/s3-dg.pdf, 2021 (**backrefpage** 79).

[137] Google Inc., *Google cloud storage documentation - object naming guidelines*, Online. Accessed 2021-02-01. https://cloud.google.com/storage/docs/naming-objects, 2020 (**backrefpage** 79).

[138] L. Orr, M. Balazinska **and** D. Suciu, *Sample debiasing in the themis open world database system (extended version)*, 2020. arXiv: 2002.09799 [cs.DB] (**backrefpage** 79).

[139] IBM Corporation, *WebSphere Application Server (Distributed operating systems) Namespace Federation*, Version 8.5.5. Online. Accessed 2021-02-01. https://www.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.base.doc/ae/cnam_federation.html, 2020 (**backrefpage** 80).

[140] The Kubernetes Authors, *Kubernetes Concepts: Namespaces*, Version 1.16. Online. Accessed 2021-02-01. https://v1-16.docs.kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/, 2021 (**backrefpage** 80).

[141] HUAWEI TECHNOLOGIES CO., LTD., *Multi-cloud container platform: Namespaces*, Issue 01. Online. Accessed 2021-02-01. https://support.huaweicloud.com/intl/en-us/usermanual-mcp/mcp_01_0025.html, 2020 (**backrefpage** 80).

[142] S. Srinivas, *An Introduction to HDFS Federation*, Online. Accessed 2021-02-01. https://blog.cloudera.com/an-introduction-to-hdfs-federation/, 2011 (**backrefpage** 80).

[143] A. Bakre, D. Krishnamurthy, K. Muthyala, C. Sharma **and** R. Talwadker, *Federated namespace of heterogeneous storage system namespaces*, Patent US10812313B2. Online. Accessed 2021-02-01. https://patents.google.com/patent/US10812313B2/en, 2021 (**backrefpage** 80).

[144] Oracle, *Java Platform Standard Ed. 7 - Package org.omg.CosNaming*, Online. Accessed 2021-02-01. https://docs.oracle.com/javase/7/docs/api/org/omg/CosNaming/package-summary.html, 2020 (**backrefpage** 80).

[145] Object Management Group, Inc., *Naming Service Specification*, Version 1.3. Online. Accessed 2021-02-01. https://www.omg.org/spec/NAM/1.3/PDF, 2004 (**backrefpage** 80).

[146] Linux man-pages Project, *Xattr - extended attributes*, Linux Programmer's Manual, Online. Accessed 2021-02-01. https://man7.org/linux/man-pages/man7/xattr.7.html, 2020 (**backrefpage** 80).

[147] A. Grünbacher, "POSIX Access Control Lists on Linux," **in***Proceedings of the FREENIX Track: 2003 USENIX Annual Technical Conference* San Antonio, TX, USA, 2003 (**backrefpage** 81).

[148] D. Rahn, *CVS log for src/sys/ufs/ufs/Attic/extattr.h*, Revision 1.2. Online. Accessed 2021-02-01. https://cvsweb.openbsd.org/src/sys/ufs/ufs/Attic/extattr.h, 2005 (**backrefpage** 81).

[149] K. Birman **and** T. Joseph, "Exploiting virtual synchrony in distributed systems," **in***Proceedings of the eleventh ACM Symposium on Operating systems principles* **volume** 21, 1987, **pages** 123–138 (**backrefpage** 85).

[150] M. Todesco, G. L. Owens, N. Bercovich, J.-S. Légaré, S. Soudi, D. O. Burge, K. Huang, K. L. Ostevik, E. B. M. Drummond, I. Imerovski, K. Lande, M. A. Pascual-Robles, M. Nanavati, M. Jahani, W. Cheung, S. E. Staton, S. Muños, R. Nielsen, L. A. Donovan, J. M. Burke, S. Yeaman **and** L. H. Rieseberg, "Massive haplotypes underlie ecotypic differentiation in sunflowers," *Nature*, **jourvol** 584, **number** 7822, **pages** 602–607, 2020.

[151] R. JESHION, "The significance of names," *Mind & Language*, **jourvol** 24, **number** 4, **pages** 370–403, 2009. DOI: https://doi.org/10.1111/j.1468-0017.2009.01367.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1468-0017.2009.01367.x. **url**: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1468-0017.2009.01367.x.

[152] J. Koetsier, *Digg pokes up its head and says, 'i'm not dead yet'*, (accessed January 6, 2019), **january** 2013. **url**: https://tinyurl.com/yam3xyp4.

[153] D. M. Ritchie **and** K. Thompson, "The unix time-sharing system," **in***ACM SIGOPS Operating Systems Review* ACM, **volume** 7, 1973, **page** 27.

[154] M. Wilkes, "A programmer's utility filing system," *The Computer Journal*, **jourvol** 7, **number** 3, **pages** 180–184, 1964.

[155] D. Reinsel, J. Gantz **and** J. Rydning, "Data age 2025: The digitization of the world from edge to core," Seagate, techreport, **november** 2018, (accessed January 6, 2019). **url**: https://tinyurl.com/y95ogat4.

[156] D. MacDonald, "Datafile: A new tool for extensive file storage," **in***Papers and discussions presented at the December 10-12, 1956, eastern joint computer conference: New developments in computers* ACM, 1956, **pages** 124–128.

[157] F. Halasz **and** T. P. Moran, "Analogy considered harmful," **in***Proceedings of the 1982 conference on Human factors in computing systems* ACM, 1982, **pages** 383–386.

[158] J. S. Shapiro, "Extracting the lessons of multics," 2004.

[159] G. Marsden **and** D. E. Cairns, "Improving the usability of the hierarchical file system," **in***Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology* South African Institute for Computer Scientists **and** Information Technologists, 2003, **pages** 122–129.

[160] W. Jones, A. J. Phuwanartnurak, R. Gill **and** H. Bruce, "Don't take my folders away!: Organizing personal information to get things done," **in***CHI'05 extended abstracts on Human factors in computing systems* ACM, 2005, **pages** 1505–1508.

[161] D. R. Karger **and** W. Jones, "Data unification in personal information management," *Communications of the ACM*, **jourvol** 49, **number** 1, **pages** 77–82, 2006.

[162] C. Jensen, H. Lonsdale, E. Wynn, J. Cao, M. Slater **and** T. G. Dietterich, "The life and times of files and information: A study of desktop provenance," **in***Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* ACM, 2010, **pages** 767–776.

[163] A. K. Karlson, G. Smith **and** B. Lee, "Which version is this?: Improving the desktop experience within a copy-aware computing ecosystem," **in***Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* ACM, 2011, **pages** 2669–2678.

[164] W. Odom, A. Sellen, R. Harper **and** E. Thereska, "Lost in translation: Understanding the possession of digital things in the cloud," **in***Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* ACM, 2012, **pages** 781–790.

[165] E. Thereska, O. Riva, R. Banks, S. Lindley, R. Harper **and** W. Odom, "Beyond file systems: Understanding the nature of places where people store their data," Microsoft Research, techreport, 2013, (accessed March 29, 2018). **url**: https://www.microsoft.com/en-us/research/wp-content/uploads/2013/02/MSR-TR-2013-26.pdf.

[166] E. Méndez **and** J. Greenberg, "Linked data for open vocabularies and hive's global framework," *El profesional de la información*, **jourvol** 21, **number** 3, **pages** 236–244, 2012.

[167] C. Bothorel, J. D. Cruz, M. Magnani **and** B. Micenkova, "Clustering attributed graphs: Models, measures and methods," *Network Science*, **jourvol** 3, **number** 3, **pages** 408–444, 2015.

[168] Z. Huo, L. Xiao, Q. Zhong, S. Li, A. Li, L. Ruan, S. Wang **and** L. Fu, "Mbfs: A parallel metadata search method based on bloomfilters using mapreduce for large-scale file systems," *The Journal of Supercomputing*, **jourvol** 72, **number** 8, **pages** 3006–3032, 2016.

[169] T. Jayalakshmi **and** C. Chethana, "A semantic search engine for indexing and retrieval of relevant text documents," 2016.

[170] N. Carvalho, H. Kim, M. Lu, P. Sarkar, R. Shekhar, T. Thakur, P. Zhou, R. H. Arpaci-Dusseau **and** I. Datos, "Finding consistency in an inconsistent world: Towards deep semantic understanding of scale-out distributed databases.," **in***HotStorage* 2016.

[171] Y. Gao, F. Song, X. Xie **and** X. Gao, "Implicit semantic text retrieval and distributed implementation for rural medical care," **in***Cloud Computing and Intelligence Systems (CCIS), 2016 4th International Conference on* IEEE, 2016, **pages** 264–267.

[172] F. J. Corbató **and** V. A. Vyssotsky, "Introduction and overview of the multics system," **in***Proceedings of the November 30–December 1, 1965, fall joint computer conference, part I* ACM, 1965, **pages** 185–196.

[173] C. Soules **and** G. Ganger, "Toward automatic context-based attribute assignment for semantic file systems," *Parallel data laboratory, Carnegie Mellon …*, **number** June, 2004. **url**: http://shiftleft.com/mirrors/www.hpl.hp.com/personal/Craig%7B%5C_%7DSoules/papers/CMU-PDL-04-105.pdf.

[174] O. Eck **and** D. Schaefer, "A semantic file system for integrated product data management," *Advanced Engineering Informatics*, **journvol** 25, **number** 2, **pages** 177–184, 2011, ISSN: 14740346. DOI: 10.1016/j.aei.2010.08.005.

[175] Y. Hua, H. Jiang, Y. Zhu **and** D. Feng, "Rapport: Semantic-sensitive Namespace Management in Large-scale File Systems," *CSE Technical reports*, 2010. **url**: http://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1123%7B%5C&%7Dcontext=csetechreports.

[176] A. Joshi, "Orion File System : File-level Host-based Virtualization,"

[177] B. Martin, "Formal concept analysis and semantic file systems," *Concept Lattices*, 2004. **url**: http://link.springer.com/chapter/10.1007/978-3-540-24651-0%7B%5C_%7D9.

[178] A. Shah, "Knowledge Management Enviroments for High Throughput Biology," **number** September, 2007.

[179] H. Reiser, "Name spaces as tools for integrating the operating system rather than as ends in themselves," 2007, (accessed October 28, 2016 via archive.org). **url**: http://www.namesys.com/whitepaper.html.

[180] I. M. Benefits, R. Current, F. Systems, D. Anyone, D. Space, I. Approach, A. Approach, D. Canonical **and** A. Comparison, "Semantic File Systems," **pages** 1–17, 2016.

[181] M. Os, O. Xtagfs, M. Os, S. Comment, M. Os, T. F. Files, S. Comments, G. Info, S. Comments, I. Macfuse, D. Xtagfs **and** R. Xtagfs, "Archive," **pages** 1–2, 2016.

[182] V. Prabhakaran, A. C. Arpaci-Dusseau **and** R. H. Arpaci-Dusseau, "Analysis and evolution of journaling file systems," *Proceedings of the International Conference on Dependable Systems and Networks*, **page** 8, 2005. DOI: 10.1109/DSN.2005.65. **url**: http://dl.acm.org/citation.cfm?id=1247360.1247368.

[183] B. Schandl, "Representing linked data as virtual file systems," *CEUR Workshop Proceedings*, **journvol** 538, 2009, ISSN: 16130073.

[184] P. Cellier, M. Ducassé, S. Ferré **and** O. Ridoux, "Formal concept analysis," *Lecture notes in computer science (R. Medina, S. Obiedkov, eds.)*, **journvol** 4933, 2008.

[185] T. Strong **and** P. Akundi, "A semantic cloud for file system annotation," **in***Proceedings of the International Conference on Semantic Web and Web Services (SWWS)* The Steering Committee of The World Congress in Computer Science, Computer Engineering **and** Applied Computing (WorldComp), 2013, **page** 24.

[186] L. Xu, Z. Huang, H. Jiang, L. Tian **and** D. Swanson, "VSFS: A searchable distributed file system," *Proceedings of PDSW 2014: 9th Parallel Data Storage Workshop - Held in Conjunction with SC 2014: The International Conference for High Performance Computing, Networking, Storage and Analysis*, **pages** 25–30, 2014. DOI: 10.1109/PDSW.2014.10.

[187] A. Leung, A. Parker-Wood **and** E. Miller, "Copernicus: A scalable, high-performance semantic file system," *University of California, Santa …*, **number** October, 2009. **url**: http://www.ssrc.ucsc.edu/papers/ssrctr-09-06.pdf.

[188] B. Schandl **and** B. Haslhofer, "The sile model - A semantic file system infrastructure for the desktop," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, **jourvol** 5554 LNCS, **pages** 51–65, 2009, ISSN: 03029743. DOI: 10.1007/978-3-642-02121-3_8.

[189] A. Shah **and** L. Caves, "ConceptOntoFs : A Semantic File System for Inferno," *1st International Workshop on Plan 9*, 2006.

[190] T. Wiki, G. P. Open **and** B. Explore, "oniony / TMSU," **pages** 1–2, 2016.

[191] K. J. Vicente, B. C. Hayes **and** R. C. Williges, "Assaying and isolating individual differences in searching a hierarchical file system," *Human factors*, **jourvol** 29, **number** 3, **pages** 349–359, 1987.

[192] T. W. Malone, "How do people organize their desks?: Implications for the design of office information systems," *ACM Transactions on Information Systems (TOIS)*, **jourvol** 1, **number** 1, **pages** 99–112, 1983.

[193] I. G. Terrizzano, P. M. Schwarz, M. Roth **and** J. E. Colino, "Data wrangling: The challenging yourney from the wild to the lake.," **in***CIDR* 2015.

[194] R. Watson, S. Dekeyser **and** N. Albadri, "Exploring the design space of metadata-focused file management systems," **in***Proceedings of the Australasian Computer Science Week Multiconference* ACM, 2017, **page** 20.

[195] P. J. Guo **and** M. Seltzer, "Burrito: Wrapping your lab notebook in computational infrastructure," **in***Proceedings of the 4th USENIX Workshop on the Theory and Practice of Provenance* **jourser** TaPP'12, Boston, MA: USENIX Association, 2012. **url**: http://dl.acm.org/citation.cfm?id=2342875.2342882.

[196] P. Macko, D. Margo **and** M. Seltzer, "Performance introspection of graph databases," **in***Proceedings of the 6th International Systems and Storage Conference* ACM, 2013, **page** 18.

[197] I. V. Balabine, R. Kandasamy **and** J. A. Skier, *File system interface to a database*, US Patent 5,937,406, **august** 1999.

[198] M. Stonebraker, P. M. Aoki, R. Devine, W. Litwin **and** M. Olson, "Mariposa: A new architecture for distributed data," **in***Data Engineering, 1994. Proceedings. 10th International Conference* IEEE, 1994, **pages** 54–65.

[199] I. V. Balabine, R. Kandasamy **and** J. A. Skier, *Database interface for database unaware applications*, US Patent 6,442,548, **august** 2002.

[200] Z. Xu, M. Karlsson, C. Tang **and** C. T. Karamanolis, "Towards a semantic-aware file store.," **in***HotOS* 2003, **pages** 181–187.

[201] A. Kashyap **and** A. Kashyap, "File system extensibility and reliability using an in-kernel database," phdthesis, Stony Brook University, 2004.

[202] J. Koren, A. Leung, Y. Zhang, C. Maltzahn, S. Ames **and** E. Miller, "Searching and navigating petabyte-scale file systems based on facets," **in***Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07* ACM, 2007, **pages** 21–25.

[203] H. H. Huang, N. Zhang, W. Wang, G. Das **and** A. S. Szalay, "Just-in-time analytics on large file systems," *IEEE Transactions on Computers*, **jourvol** 61, **number** 11, **pages** 1651–1664, 2012.

[204] Z. Xu, M. Karlsson, C. Tang **and** C. Karamanolis, *Semantic file system*, US Patent 7,617,250, **november** 2009.

[205] N. Murphy, M. Tonkelowitz **and** M. Vernal, *The design and implementation of the database file system*, 2002.

[206] K.-Y. Whang **and** R. Krishnamurthy, "The multilevel grid file-a dynamic hierarchical multidimensional file structure.," **in***DASFAA* World Scientific, **volume** 1991, 1991, **pages** 449–459.

[207] H. P. Kumar, C. Plaisant **and** B. Shneiderman, "Browsing hierarchical data with multi-level dynamic queries and pruning," *International journal of human-computer studies*, **jourvol** 46, **number** 1, **pages** 103–124, 1997.

[208] F. Van Ham **and** J. J. van Wijk, "Beamtrees: Compact visualization of large hierarchies," *Information Visualization*, **jourvol** 2, **number** 1, **pages** 31–39, 2003.

[209] J. MacCormick, N. Murphy, M. Najork, C. A. Thekkath **and** L. Zhou, "Boxwood: Abstractions as the foundation for storage infrastructure.," **in***OSDI* **volume** 4, 2004, **pages** 8–8.

[210] G. Marchionini, "Exploratory search: From finding to understanding," *Communications of the ACM*, **jourvol** 49, **number** 4, **pages** 41–46, 2006.

[211] S. Brandt, C. Maltzahn, N. Polyzotis **and** W.-C. Tan, "Fusing data management services with file systems," **in***Proceedings of the 4th Annual Workshop on Petascale Data Storage* ACM, 2009, **pages** 42–46.

[212] A. W. Leung, M. Shao, T. Bisson, S. Pasupathy **and** E. L. Miller, "Spyglass: Fast, scalable metadata search for large-scale storage systems.," **in***FAST* **volume** 9, 2009, **pages** 153–166.

[213] K. Ren **and** G. A. Gibson, "Tablefs: Enhancing metadata efficiency in the local file system.," **in***USENIX Annual Technical Conference* 2013, **pages** 145–156.

[214] S. Niazi, M. Ismail, S. Haridi, J. Dowling, S. Grohsschmiedt **and** M. Ronström, "Hopsfs: Scaling hierarchical file system metadata using newsql databases.," **in***FAST* 2017, **pages** 89–104.

[215] R. V. H. Van Staereling, R. Appuswamy, D. C. van Moolenbroek **and** A. S. Tanenbaum, "Efficient, modular metadata management with loris," **in***Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on* IEEE, 2011, **pages** 278–287.

[216] R. Appuswamy, D. van Moolenbroek **and** A. Tanenbaum, "Loris - a redundant array of independent physical layers," English, **in***Proceedings of the sixteenth annual conference of the Advanced School for Computing and Imaging (ASCI'10)* Advanced School for Computing **and** Imaging (ASCI), 2010.

[217] R. Appuswamy, "Building a file-based storage stack: Modularity and flexibility in loris," 2014.

[218] M. Harlan, "JOIN FS : a Semantic File System for Embedded Systems," 2011.

[219] C. Min, S. Kashyap, B. Lee, C. Song **and** T. Kim, "Cross-checking Semantic Correctness : The Case of Finding File System Bugs," *Sosp '15*, **pages** 361–377, 2015. DOI: 10.1145/2815400.2815422.

[220] P. Open **and** B. Explore, "Fuse::TagLayer," **pages** 1–2, 2016.

[221] J. Chou, "FindFS - Adding Tag-Based Views to a Hierarchical Filesystem by," **number** August, 2015.

[222] B.-h. Ngo, C. Bac, B.-h. Ngo, C. Bac **and** F. Silber-chaussumier, "To cite this version : Integrating Ontology into Semantic File Systems," 2015.

[223] B. Gopal, "Integrating Content-Based Access Mechanisms with Hierarchical File Systems 1 Introduction 2 The Design of the HAC File System,"

[224] G. Beydoun, "Formal concept analysis for an e-learning semantic web," *Expert Systems with Applications*, **jourvol** 36, **number** 8, **pages** 10 952–10 961, 2009, ISSN: 09574174. DOI: 10.1016/j.eswa.2009.02.023.

[225] D. Gifford, P. Jouvelot, M. Sheldon, J. O. Toole, D. K. Gifford, M. A. Sheldon **and** J. W. O. Toole, "13 . Paper : Semantic File Systems Semantic File Systems," *Computer*, **pages** 16–25, 2006.

[226] S. L. Hyvonen **and** S. Louise, "Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.," *Dissertation*, **number** May, 2004.

[227] J. Siekmann, "Formal Concept Analysis," **pages** 0–45,

[228] T. Strong **and** P. Akundi, "A Semantic Cloud for File System Annotation," **pages** 1–4,

[229] R. Escriva **and** E. Gun Sirer, "The Design and Implementation of the WAVE Transactional File System," **number** February, **pages** 1–14, 2015. arXiv: 1509.07821. **url**: http://arxiv.org/pdf/1509.07821v1.pdf.

[230] B. Agee, C. Daly **and** J. Brown, "Ubuntu A tag-based filesystem for ubuntu," **pages** 1–3, 2016.

[231] D. K. Gifford, P. Jouvelot, M. a. Sheldon, J. W. O'Toole, P. Jouvelotl, M. a. Sheldon **and** W. J. O'Toole, Jr, "Semantic file systems," *ACM SIGOPS Operating Systems Review*, **jourvol** 25, **number** 5, **pages** 16–25, 1991, ISSN: 01635980. DOI: 10.1145/121133.121138. **url**: http://portal.acm.org/citation.cfm?doid=121133.121138%7B%5C%%7D5Cnhttp://dl.acm.org/citation.cfm?id=121133.121138.

[232] V. U. Guide, "Archive," **pages** 1–2, 2016.

[233] X. Liu, G. Niv, P. Shenoy, K. Ramakrishnan **and** J. Van der Merwe, "The case for semantic aware remote replication," *Proceedings of the second ACM workshop on Storage security and survivability - StorageSS '06*, **page** 79, 2006. DOI: 10.1145/1179559.1179575. **url**: http://dl.acm.org/citation.cfm?id=1179559.1179575.

[234] R. M. Needham **and** A. D. Birrell, "The cap filing system," **in***ACM SIGOPS Operating Systems Review* ACM, **volume** 11, 1977, **pages** 11–16.

[235] R. Walton, "Searching high and low: Faceted navigation as a model for online archival finding aids (a literature review).," *Journal for the Society of North Carolina Archivists*, **jourvol** 12, 2015.

[236] H. C. Huurdeman, M. L. Wilson **and** J. Kamps, "Active and passive utility of search interface features in different information seeking task stages," **in***Proceedings of the 2016 ACM on Conference on Human Information Interaction and Retrieval* ACM, 2016, **pages** 3–12.

[237] O. van Rest, S. Hong, J. Kim, X. Meng **and** H. Chafi, "Pgql: A property graph query language," **in***Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems* ACM, 2016, **page** 7.

[238] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer **and** A. Taylor, "Cypher: An evolving query language for property graphs," **in***Proceedings of the 2018 International Conference on Management of Data* ACM, 2018, **pages** 1433–1445.

[239] R. Angles, M. Arenas, P. Barcelo, P. Boncz, G. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. Sequeda **andothers**, "G-core: A core for future graph query languages," **in***Proceedings of the 2018 International Conference on Management of Data* ACM, 2018, **pages** 1421–1432.

[240] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, **jourvol** 1, **number** 1, **pages** 269–271, 1959.

[241] M. Conover, "Analysis of the windows vista security model," *Symantec Advanced Threat Research*, 2006.

[242] M. J. Spier, T. N. Hastings **and** D. N. Cutler, "An experimental implementation of the kernel/domain architecture," **in***Proceedings of the Fourth ACM Symposium on Operating System Principles* **jourser** SOSP '73, New York, NY, USA: ACM, 1973, **pages** 8–21. DOI: 10.1145/800009.808043. **url**: http://doi.acm.org/10.1145/800009.808043.

[243] L. Rosenfeld **and** P. Morville, *Information architecture for the world wide web.* " O'Reilly Media, Inc.", 2002.

[244] D. Beaver, S. Kumar, H. C. Li, J. Sobel, P. Vajgel **andothers**, "Finding a needle in haystack: Facebook's photo storage.," **in***OSDI* **volume** 10, 2010, **pages** 1–8.

[245] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang **andothers**, "F4: Facebook's warm blob storage system," **in***Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation* USENIX Association, 2014, **pages** 383–398.

[246] D. Borthakur, *Rocksdb: A persistent key-value store*, 2014.

[247] H. Lim, B. Fan, D. G. Andersen **and** M. Kaminsky, "Silt: A memory-efficient, high-performance key-value store," **in***Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* ACM, 2011, **pages** 1–13.

[248] M. K. McKusick, W. N. Joy, S. J. Leffler **and** R. S. Fabry, "A fast file system for unix," *ACM Transactions on Computer Systems (TOCS)*, **jourvol** 2, **number** 3, **pages** 181–197, 1984.

[249] M. Cao, S. Bhattacharya **and** T. Ts'o, "Ext4: The next generation of ext2/3 filesystem.," **in***LSF* 2007.

[250] G. A. I. Barnard **and** L. Fein, "An information filing and retrieval system for the engineering and management records of a large-scale computer development project," English, *American Documentation (pre-1986)*, **jourvol** 9, **number** 3, **page** 208, **july** 1958, Copyright - Copyright Wiley Periodicals Inc. Jul 1958; Last updated - 2010-08-27. **url**: http://prx.library.gatech.edu/login?url=https://search.proquest.com/docview/195441816?accountid=11107.

[251] Amazon Web Services, *Editing Object Metadata*, Online. Accessed 2021-02-01. https://docs.aws.amazon.com/AmazonS3/latest/user-guide/add-object-metadata.html, 2021.

[252] R. Arpaci-Dusseau **and** A. Arpaci-Dusseau, "Naming and binding of objects," **in***Operating Systems: Three Easy Pieces* 2021, Chapter 41. **url**: http://pages.cs.wisc.edu/~remzi/OSTEP/file-ffs.pdf.

[253] R. Daley **and** P. Neumann, "A general-purpose file system for secondary storage," **in***Proceedings of the November 30–December 1, 1965, fall joint computer conference, part I* ACM, 1965, **pages** 213–229.

[254] H. Sim, A. Khan, S. S. Vazhkudai, S. Lim, A. R. Butt **and** Y. Kim, "An integrated indexing and search service for distributed file systems," *IEEE Transactions on Parallel and Distributed Systems*, **jourvol** 31, **number** 10, **pages** 2375–2391, 2020. DOI: 10.1109/TPDS.2020.2990656.

[255] V. Atlidakis, J. Andrus, R. Geambasu, D. Mitropoulos **and** J. Nieh, "Posix abstractions in modern operating systems: The old, the new, and the missing," **in***Proceedings of the Eleventh European Conference on Computer Systems* **jourser** EuroSys '16, London, United Kingdom: Association for Computing Machinery, 2016, ISBN: 9781450342407. DOI: 10.1145/2901318.2901350. **url**: https://doi.org/10.1145/2901318.2901350.

[256] N. Albadri, R. Watson **and** S. Dekeyser, "Treetags: Bringing tags to the hierarchical file system," **in***Proceedings of the Australasian Computer Science Week Multiconference* **jourser** ACSW '16, Canberra, Australia: Association for Computing Machinery, 2016, ISBN: 9781450340427. DOI: 10.1145/2843043.2843868. **url**: https://doi.org/10.1145/2843043.2843868.

[257] A. Parker-Wood, D. D. E. Long, E. Miller, P. Rigaux **and** A. Isaacson, "A file by any other name: Managing file names with metadata," **in***Proceedings of International Conference on Systems and Storage* **jourser** SYSTOR 2014, Haifa, Israel: Association for Computing Machinery, 2014, **pages** 1–11, ISBN: 9781450329200. DOI: 10.1145/2611354.2611367. **url**: https://doi.org/10.1145/2611354.2611367.

[258] P. H. Lensing, T. Cortes **and** A. Brinkmann, "Direct lookup and hash-based metadata placement for local file systems," **in***Proceedings of the 6th International Systems and Storage Conference* **jourser** SYSTOR '13, Haifa, Israel: Association for Computing Machinery, 2013, ISBN: 9781450321167. DOI: 10.1145/2485732.2485741. **url**: https://doi.org/10.1145/2485732.2485741.

[259] I. Beckwith, "Id3fs," **url**: https://erislabs.net/ianb/projects/id3fs/id3fs-index.html.

[260] H. Kang **and** B. Shneiderman, "Mediafinder: An interface for dynamic personal media management with semantic regions," **in***CHI '03 Extended Abstracts on Human Factors in Computing Systems* **jourser** CHI EA '03, Ft. Lauderdale, Florida, USA: Association for Computing Machinery, 2003, **pages** 764–765, ISBN: 1581136374. DOI: 10.1145/765891.765977. **url**: https://doi.org/10.1145/765891.765977.

[261] C. Lee, D. Sim, J. Hwang **and** S. Cho, "F2fs: A new file system for flash storage," **in***13th {USENIX} Conference on File and Storage Technologies ({FAST} 15)* 2015, **pages** 273–286.

[262] M. Rosenblum **and** J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems (TOCS)*, **jourvol** 10, **number** 1, **pages** 26–52, 1992.

[263] J. H. Saltzer, "Naming and binding of objects," **in***Operating Systems: An Advanced Course* R. Bayer, R. M. Graham **and** G. Seegmüller, **editors**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, **pages** 99–208, ISBN: 978-3-540-35880-0. DOI: 10.1007/3-540-08755-9_4. **url**: https://doi.org/10.1007/3-540-08755-9_4.

[264] F. Revol, "Universal file system extended attributes namespace," **in***International Conference on Dublin Core and Metadata Applications* 2011, **pages** 69–73.

[265] D. Kasatkin **and** Z. Miriam, "Integrity measurement architecture (ima)," 2021. **url**: https://sourceforge.net/p/linux-ima/wiki/Home/.

[266] G. Linux, "How selinux controls file and directory accesses," 2020. **url**: https://wiki.gentoo.org/wiki/SELinux/Tutorials/How_SELinux_controls_file_and_directory_accesses.

Appendix A

Supporting Materials

This would be any supporting material not central to the dissertation. For example:

- additional details of methodology and/or data;
- diagrams of specialized equipment developed.;
- copies of questionnaires and survey instruments.

Section A.1 and Section A.2 are write-ups that I did originally as part of preparing this thesis proposal. They are included as background and I expect to remove them prior to submission.

A.1 One-Page Proposal: Evolution

File systems have evolved slowly: development approaches, operating systems interaction, and interfaces reflect decisions made decades ago. However, scale, technology, and systems have evolved. Past decisions reflect appropriate solutions that no longer fit the modern world. File systems must evolve to meet modern demands, utilize modern technologies, and encourage further innovation.

MIS *►I think the messaging is right here, but the order is wrong: That is you start with FS evolution, but if the world were static, we wouldn't care about FS evolution. So I think the motivation is that data has changed dramatically in the past 40 years: we generate 10^x times more data each year; we have rich data (audio, video, VR, etc). However, file system s are largely unchanged (slight exaggeration).* ◄

MIS *►I think I would next say something to the effect of, "My dissertation will examine N ways in which file systems must evolve, demonstrating both why and how*

they must do so. Then list the N things and go into the next paragraph. ◀ In the early 1990s file system caching underwent rapid transformation, with tight integration between the virtual memory manager and the file system: cached files were now mapped. I/O operations from applications were thus satisfied using the memory mapping (MIS ▶*Expand to what this means; using standard memory copying library calls (which, in turn, means accessing file data via load and store instructions). There are some/many who would argue that mmap cannot possibly make it faster to access data – keith is one such person – he fundamentally doesn't believe it, even after reading sasha's post carefully.*◀). This greatly improved performance and utilization of memory. File systems could thus handle fixed (page size) operations, with needed data faulted into the cache in multi-page unities that were highly efficient for storage devices. The introduction of persistent memory MIS ▶*Over the past N years, the industry has seen the emergence of persistent memory accessible from the memory bus –* ◀ in this environment required a new change, the "direct access" model, which continued to use the page based management model with direct mapping. However, this model no longer makes sense and has interesting implicit costs associated with it because small objects (files) in persistent memory still consume blocks of storage, which is inefficient, and the I/O patterns of applications, which provide insight into the structure of the data within the files, is ignored. The DaxFS project builds upon my prior work with persistent memory and combines it with work I have done to find useful insights in recognizing and using implicit object structures to provide efficient storage and sub-block sized object efficiencies, including deduplication and "patching support". MIS ▶*At this point, the key issue is not so much how it builds upon and/or relates to your own work but precisely what DaxFS is designed to do/address.* ◀

Modern production file systems are typically specialized kernel-level software. MIS ▶*I would just say, "File systems are most frequently implemented as part of the operating system, running at supervisor privilege." Or something like that. It's not a development model – it's a deployment model.* ◀ This development model hobbles innovation and exploration because it relegates user-mode file systems to specialized use, such as network file systems, or toy examples. There is no fundamental reason for this: software does not run faster in kernel mode than it does in user mode. Indeed, most operating systems make choices that make the fundamental operation of copying data between memory locations slower in kernel mode than user mode. Prior work has explored techniques for improving this performance using in-kernel techniques such as optimized communications and specialized kernel extensions as well as a single development model for user and kernel file systems. I propose finding mechanisms that avoid kernel interaction at all, as a form of "kernel

bypass" reminiscent of how file systems are implemented in micro-kernel systems, with the goal being to leverage the simpler user mode development environment to foster further innovation building file systems. MIS ►*I think this requires more precision to differentiate from the gazillion papers on kernel bypass.* ◄

File systems have been using a simple naming scheme with nominal meta-data MIS ►*More precise "hierarchical namespace" – the meta data issue that the current meta data is designed for use by the FILE SYSTEM not people.*◄, hearkening back to a time when users did not store related files in unrelated locations. This simple model has served us well but has serious limitations in modern storage systems. I have identified specific issues MIS ►*Vague – be more precise about what you have identified.*◄ related to capturing data relationships in a fashion that permits association of related data, including files, stored in unrelated locations, such as storage systems with different characteristics and object stores. This work is to construct a system that captures activity context MIS ►*Which is not defined...*◄ and then permits me to create namespace views driven by those activity contexts to show data relationships. This will allow me to answer the question of whether or not such a system can simplify data location, reduce data duplication, and ensure efficient use of storage by allowing related data to be stored in unrelated storage locations. MIS ►*What is the ultimate problem you are trying to solve? People can't find their data! Just say that – then say that you have concrete ideas about how to make it easier – and it's all about activity context, which is ….. and enables …* ◄

Thus, the goal of my thesis MIS ►*Think about this in terms of a thesis statement you will ultimately be able to defend. E.g., Adding features X, Y, and Z to file systems enables them to better support modern file system usage, such as ….* ◄ is to revisit old decisions in light of new information, find old or new paradigms for storing and organizing information, with a goal towards simplifying development of new file systems, as well as providing greater utility and performance.

A.2 One-Page Proposal: Kwishut

Existing storage systems often operate in isolation. MIS ►*Compare this opening to the following, 'users today store and access data from myriad different locations'*◄ This does not match the usage or expectations of users: large static files may be stored in an optimized storage silo, while smaller dynamic files constructed from those large files may be stored in a collaborative

storage silo, such as Google Docs or Excel Online. This situation is exacerbated by a naming convention that either uses proprietary meta-data management mechanisms or forces users to embed critical meta-data into the file and path name structure. Prior work, such as Placeless and Ground, have attempted to address aspects of this MIS ►*But what is "this" ?*◄ but have fallen short: Placeless fails to capture the "activity context" MIS ►*Not defined*◄ information so critical to tying related data objects together and Ground identifies interesting research questions without addressing them. MIS ►*Here is what I think this paragraph is saying, "Users today store and access data in myriad different places. This distributed storage makes finding objects difficult. I claim that users possess so much data with many complex and varying relationships that the inability to find things is due to a fundamental mismatch between the way users want to interact with their data and the way that systems enable them to interact with them. Kwishmut is the name we give to an architecture designed to address this fundamental mismatch.* ◄

Kwishut sets out to extend this prior work and address key research questions that we have identified: the cost and benefits of: (1) explicitly separating meta-data management, including naming, from storage silo location and management; (2) securing, protecting the privacy and ensuring the integrity of meta-data; and (3) constructing usable namespace views, including traditional hierarchical, search driven, and novel data visualization strategies. MIS ►*It's not entirely clear that we are at a point to cast this in terms of cost/benefit. That might be a way to evaluate it, but I'm not sure that's the question we are trying to answer – I think you are claiming (i.e., a thesis statement) might be of the form that "Explicit separation of meta data and naming, enabling the construction of per-user namespaces, and capturing information about the context in which data is constructed and accessed are critical to supporting modern data management needs." [that's not quite right, but it's close.]* ◄

This system involves the design, development, and evaluation of key components: (1) a meta-data service that extends the capabilities of existing meta-data services and permits capturing the richer meta-data Kwishut enables; (2) a namespace service that utilizes the meta-data services to realize usable cross-storage silo namespace servers; (3) activity monitors that record specific activity context on local devices and store that in a meta-data service so that this rich context is available to the namespace services; (4) attribute services that extract attribute information from existing storage silos and store that in a meta-data service; and (5) an update notification service that permits registration for specific events and notification when those events are observed. MIS ►*Is there an MVP subset? (E.g., could you design an architecture that includes notification, necessary for certain applications, but*

focus on those apps that do not require it?) ◄

This proposal is quite broad and consists of no working software at the present time. To achieve this I propose taking the initial architecture and constructing a design based upon this architecture. To evaluate the design, I propose choosing at least two distinct storage silos and at least one target operating system. To the extent possible, initial implementation would focus on combining existing components as much as possible. For example, Using MinIo and Sparkle Share as storage silos, with Windows Cloud Sync would provide documented and generally well-understood technologies on which to construct these services. The Meta-data server can be constructed using one of the available key-value stores (e.g., WiredTiger). The Notification service could start with Emitter. Initial activity context work can be built using eBPF or other inbuilt tracing mechanisms. This leaves how to build the attribute services as an open area to be further refined. MIS ►*I think the message here is, "At present we have only a high level design of the system. We can develop a prototype implementation by leverage existing technology for most of the components, implementing only those parts necessary to address the key research questions. For example, we do not need to build a new file system, but can leverage existing ones such as … Nor do we need to implement a meat-data store. Instead we need to develop techniques to extract meta-data from data and then store that meta-data in off the shelf technology, such as …*◄

The key research questions are: (1) what benefits can be realized in terms of using cross-silo meta-data to create namespaces based upon logical associations rather than traditional silo/path/name style organization; (2) what are the costs related to these; (3) what are the security issues we can identify and how do we address them in Kwishut; (4) how can this system be used to improve data governance; and (5) can we enable construction of innovative new data visualizations of cross-silo data that are not presently possible.

A.3 HotOS 2019

A.3.1 Abstract

Rumors of the demise of the hierarchical file system namespace have been greatly exaggerated. While there seems to be wide spread agreement that most users make no use of the hierarchical name space, we continue to use it both as the underlying storage structure and as the default user interface. To compensate for this mismatch between the native organization and the

way users interact with their data, we have produced myriad search tools atop the file system. This approach, however, has some limitations. In particular, we focus on the absence of generalized relationships as first class entities.

The hierarchical namespace elevates the *contains* relationship above all else. Applications elevate the *was created by me* relationship. These are only two particular relationships among many; items accessed around the same time share a temporal relationship; attachments to email messages share a relationship; a collection of documents, email message, notes, etc. form another relationship. We claim that elevating arbitrary and generalized relationships as first class file system elements provides a better user experience and leads to entirely different implementation approach.

A.3.2 A Modest Proposal

Our proposed file system focuses on *relationships* between our files. We use an analogy between social graphs and file systems to explore this approach. Facebook's graph is a collection of typed objects (e.g., users, actions, places) and associations (e.g., friend, authored, tagged). File system objects map to users and files; *contains* is, perhaps, the only association captured in a file system. For the rest of this discussion, we treat directories as the embodiment of the *contains* relationship, not objects.

We consider two strawman implementations for elevating relationships to first class file system objects.

File System as a Graph Database

In considering a graph based file system, first we consider implementing a file system in a graph database, of which there are many (§A.3.4). Their primary focus is the storage of graph-structured *data*. Our focus is in the use of graph-structured data as critical meta-data inside of a storage system. As such, there is a mismatch in design targets between a file system and existing graph databases: nodes in graph databases are small; nodes in file systems are large. Graph databases tend to favor a navigation-based API; file systems need a point query and search API. Graph databases assume that attributes and relationships are provided; file systems will frequently derive attributes and relationships. These differences suggest to us that existing graph databases are not suitable as the basis for file systems. Nonetheless, we encourage others to consider such an arrangement, should they have compelling reason to do so.

File System as Social Network

Next, we consider implementing a file system in the same way Facebook implements their social network graph. Facebook's original implementation stored the social graph in MySQL, queried it from PHP, and cached the results in memcache. More recently, Facebook introduced Tao, which is a service that more directly implements the fundamental objects and relationships that comprise the social graph [51]. While Tao is specifically designed to support the widely distributed, replicated, and rapidly changing social network scenario, it provides the starting point for conceptualizing a data model premised on the primality of relationships. Tao stores both objects and associations in a MySQL database and presents the graph abstraction via Association and Query APIs in the caching layer. Is this a viable structure for a file system?

Unlike objects in Facebook, files are large. Although prior work has considered using relational database [52] and other index-based structures [53] to store files, the community seems to have concluded that such storage is not ideal. We agree, suggesting that an RDBMS is not the desired storage system.

What about relationships? Is it appropriate to use one persistent representation (e.g., a relational one) and a second memory representation (e.g., a graph-structured on) or should we use a single graph-structured representation both in persistent store and in-memory. We propose the latter for two reasons. First, the rumored era of non-volatile main memory seems to be around the corner, so a modern file system design should embrace a single representation. Second, while it is reasonable for Facebook to construct the entire graph in a distributed pool of main memory, file systems must work on a more limited scale and therefore cannot ensure that the realized graph structure will fit in main memory.

As neither strawman design seems suitable for our relationship-centric file system, we present a new model and file system design.

A.3.3 Graph FS Model

We set out a basic description of our core objects in Table A.1 and a demonstrative set of example relationships in Table A.2. We do not consider either of these to be exhaustive, but rather propose them as an initial basis for discussion. The presented model can encompass the functionality of the existing hierarchical file system model.

Every file has a unique identifier, such as a **UUID**, similar to an inode

Term	Definition
file	Uniquely identified storage unit
relationship	Directional file association
labels	A binary attribute, e.g., executable
property	Key/Value attribute

Table A.1: Graph File Systems Terminology

number or object ID. We do not rely upon *names* as they are simply mutable properties.

Relationship	Description
similar	Similarity measure, e.g., [54]
precedes	temporal relationship (e.g., versioning)
succeeds	temporal relationship (e.g., versioning)
contains	directory/file relationship
contained by	directory/file relationship
derived from	provenance (e.g., .o to .c)

Table A.2: Graph File System Relationship Examples

A *relationship* is a directional association between two files. We expect there to be far fewer relationships than files, though many more *instances* of relationships (i.e., the number of edges in our graph exceeds the number of vertices). Relationships may be either uni-directional (e.g., derived from) or bi-directional (e.g., similar). Table A.2 provides a set of sample relationships; the universe of relationships is extensible. As in RDF, relationships are triples: two files and the relationship.

As files have attributes in a conventional file system, both files and relationships have attributes in a graph file system. A *label* is a simple binary attribute (e.g., executable), while a *property* is an arbitrary name/value pair, much like an extended attribute, but they are native to the model, not an afterthought.

Interface

One of the lessons from Plan 9 is that everything can be represented as a file [55]; we expect to continue with this paradigm as it has served us well over the years. While we generally think of files as a blob of *persistent* data, in fact it is useful to think of them as abstract *generators* of byte stream

data. This fits well with our model of separating namespace from storage; how the storage providers return data to us is orthogonal to the namespace we use to retrieve it. For example, the *procfs* file system creates a synthetic namespace and supports I/O operations for reading and modifying data contents of the pseudo-files.

From the namespace perspective, our file system must support operations that manipulate that namespace. This includes the ability to create files, relationships, relationship instances *between* files, labels, and properties. Similarly, we need the ability to remove each of these.

Our model is simple, yet powerful. It captures interesting concepts such as versioning, using relationships such as *precedes* and *succeeds*, and provenance, using relationships around derivation and use, and application specific relationships, such as *indexes* so a database system can expose the relationship between its primary data and the corresponding index files. Although relationships are binary, we can create clusters of related files by asking for all the vertices connected by a specific relationship.

Where do relationships, labels, and properties come from? We identify at least the following five sources: 1) the system itself will generate traditional attributes (e.g., *size*, *read time*) and some relationships (e.g., contains); 2) tools that extract meta-data from different file types [56], [57] will produce more attributes; 3) applications will generate both attributes and relationships; 4) users may generate attributes and relationships, although history suggests that asking users to annotate data is a losing proposition [58]; and 5) kernel extensions, e.g., provenance tracking systems [59] will generate attributes and relationships.

Several interesting possibilities emerge from this design. Hard links are multiple *name* properties attached to the same file, potentially in different namespaces. Soft links are a relationship between two names. The system can capture relationships that extend beyond the file system. For example, the *derived from* relationship from Table A.2 might describe a file that came from a particular email or web site.

Operation	Description
create	Insert new file into graph
relate	Insert new edge into graph
label	Insert new labels
set	Insert new properties
remove	Remove something from graph

Table A.3: Graph File Systems Operations

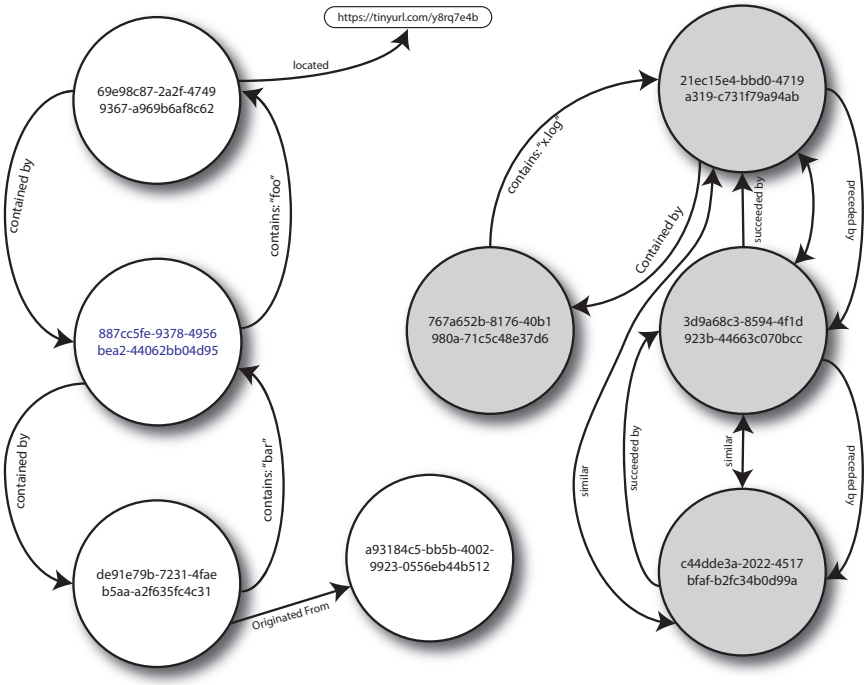

Figure A.1: Graph File System

Figure A.1 provides a simplified visualization of our graph file system model. Our inclusion of disjoint graphs captures the notion that the system naturally supports multiple namespaces, implemented as disconnected graph components.

A.3.4 Aspects of Implementation

In the absence of space to provide a full implementation, we offer a few strategies that make a graph file system both feasible and novel. The underlying storage structure for files is essentially an object store [60] and attribute storage is largely a solved problem (although the last time one of the authors said that, her colleague disagreed [61]), so we focus on fast and efficient graph storage and query.

Today's systems either provide graph storage [62]–[64] or graph processing [65]–[71], but a graph file system needs a high performance, space-efficient, mutable and queryable native graph representation. We have found that mutable compressed sparse row representations [72] meet all these requirements (we

used them as the query and storage mechanism in the SHEEP graph partitioner [73]). Just as high performance key/value stores are considered reasonable implementation strategies for attribute storage and management in file systems, similarly efficient structures supporting graph storage and management should be adopted in file systems as well.

Search

The driving force behind our graph file system design is to provide the infrastructure to make it easier for users to find data. Users do not navigate to data, they *search* for it, so we consider more effective search models to further motivate the graph file system.

We observe that there are two different models of "search": application search and user search. Applications need to be able to open files quickly using a *key*. For example, both NFS [74] and AFS [75] use the file system *inode number* as their mechanism for identifying the specific file or directory being accessed, because it is fast, avoiding a costly namespace traversal. Similarly, NTFS supports the ability of applications to open a file by identifier [76]. They did this to support their implementation of the Apple File Protocol ("service for Macintosh") but has subsequently been used for other uses. Indeed, it has been further extended to permit files to be opened by an application-defined identifier (a UUID); Microsoft continues to support file IDs in ReFS [77]. The Google File System [78] observation was similar: applications can use keys to find their files.

Modern applications tend to either create files that they use internally, often going to great lengths to hide their location from the user; or maintain a list of recently used items with a full path name, which breaks when the path changes, even if the file did not change. A key interface for applications better fits this usage model. Thus, a "search by key" interface is sufficient.

The more challenging problem is user-focused search. For human users, we want to enable a model like the *memex* [2]: "A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory."

The HCI community has a long history researching more effective search, including such efforts as SIS [79] and faceted search [80]–[85]. Critical to this work is the idea that search is most effective when *not* bound to a specific taxonomic order — very much the opposite of today's hierarchical search model, which enforces a rigid order on the structure of information.

How does a graph file system then enable modern search? First, support

for a broad and extensible set of attributes and relationships brings search engine technology to bear in the service of file systems. There is some irony that the success of web search, and in particular the primacy of relationships in those algorithms [31], has had virtually no impact on how we find our own local data. Second, the generalized graph structure, which no longer elevates any single organzation gets rid of the *specific taxnomic order* that HCI researchers determine to be counterproductive. Third, although some degree of temporal query is possible using `find`, its interface is not especially accessible to the typical user, and it requires a series of manual operations to express natural queries such as "Show me the documents I wrote last summer after I got back from my Amazon rafting trip."

Our goal is not to specify the entire range of searches that can be realized, but rather to explore file system structures that enable the creation and mining of relationships to help users to find relevant data.

Related Work

The need for better name spaces in file systems is hardly a new topic, with many solutions being proposed and implemented over the years.

Search utilities are successors to the permuted index program. They permit us to find files based on *content* and *attributes*. MacOS X has *spotlight*, which provides an extensible, index-driven search service [86]. Similarly, Windows offer a similar extensible service [87]. These enable searching based upon attributes, e.g., file suffix, date, size, etc., and context-sensitive content, e.g., music files by artist, composer, song title, or even *rights*, but limited, if any, ability to search by relationship.

Tag Systems were an early approach to improving hierarchical file systems searchability [88]–[99]. Automatic tagging systems have become a more common approach here as manual tagging by users has proven to be impractical [56], [58]. The addition of *semantic* information [92], [94], [95], [100]–[115] is useful but falls short of addressing the fundamental need to understand data relationships, because like more conventional systems, these semantically-aware approaches still focus on the file, not on the relationships between the files. As such, they are simply an add-on to the hierarchical model, not a replacement for it. Such approaches can provide useful functionality in our graph file system model as well. Indeed, we even pointed this out (perhaps subconsciously) in prior work when we said "How many of them [files] are *related* to each other?" [emphasis added] [116] .

Files are rich with relationships. However, these relationships are not limited to what the file system can "see". Narrowing our vision to the

closed pool of file system relationships hobbles our ability to capture them. For example, the obvious relationship between a file and the e-mail from whence it originated is not exploitable in any system of which we are aware. The academic papers we generate refer to other papers. An enlightened document application would provide an identifier that can be used to find the corresponding paper - a *refers to* relationship, whether on our local system or elsewhere. Of course, in the current model, we likely can't recall where we stored it when we downloaded it. As we create new works, we refer to older works — our own documents, web pages, Jupyter notebooks, spreadsheets, pictures, etc. Capturing these relationships permits us to reconstruct the process taken to produce an output. This is the fundamental problem that the provenance community has been addressing, but few systems [59], [117] demonstrate an understanding of the role our file systems play in making this possible.

Versioning is a feature that continues to reappear in various guises. This is simply one example of temporal locality; a relationship that we have not yet deeply mined. While it is now common practice for individual applications to "remember" the last few files you have accessed, there are few cross-application examples. In lieu of the right tools, users invent creative solutions. For example, one of the authors *attaches* documents to e-mail immediately after reviewing them specifically to establish temporal relationship. The ability to establish temporal relationships across applications should provide powerful capabilities.

We do not know which relationships are useful. One of the hallmarks of good file systems design over the decades has been *not* to impose a specific restricted model on what files can be — we leave that to databases. We do not intend to establish a definitive set of relationships any more than we focus on defining the structure of file contents. However, we encourage others to explore this area, encourage best practices, and build tools that produce and use such relationships, leaving the storage and retrieval of relationships to the file system.

Much of the raw data that applications generate would be better *not* injected into the hierarchical name space: the location of our personal email database, financial software files, binaries, temporary files, etc. Their presence in the name space clutter it and make our existing brute force search slower yet no more useful.

Application programs benefit being unfettered from the hierarchical name space [78], both in terms of their efficiency as well as the benefits of *not* commingling the private files of individual applications — but the hierarchical file system requires they be stored somewhere within its domain. Applications

routinely hide most of their files in out of the way locations, as they are only useful to the application itself. Thus we end up with "System Volume Information" and ".ssh" and a myriad of obscure locations where applications hide data from us. This is a side effect of the name space model we have used for the past 50 years.

Prior attempts to address this have done so within the confines of a narrow perspective of what is needed to fully enable the ability of us to find our data. The HCI community have been poking at the edges of this problem for decades as well — observing the frailties of the hierarchical model and suggesting alternatives [91], [92], [104], [109], [118]–[124].

Our graph file system permits them to escape the existing paradigm *without* giving up support of existing applications.

A.4 HotOS 2021

A.4.1 Absract

File systems use names to store and retrieve file data. A system can use a machine-generated name to locate an object, while a user needs a semantically meaningful name to locate it. Most systems either lump these two names into one, or tie them together within a particular storage silo, e.g., a file system, Google docs, or a cloud store. As a result, it is difficult to search and navigate objects across silos and view them within a meaningful context that is not necessarily tied to file names.

We propose Nirvana, an architecture decoupling system-level *location names* from human-friendly *context/semantic names*. Nirvana generalizes the definition of a semantic name to be a set of metadata attributes, which may or may not include a traditional user-friendly name. We explain how Nirvana can enhance user experience and describe system support needed to realize it.

A.4.2 Introduction

Today's file systems provide two primary functions: a way to store chunks of data and names that provide users with the familiar metaphor of a file cabinet (i.e., folders and documents). Much has changed since we adopted this design. Now most data does not reside on local file systems because instead data is distributed across myriad services such as Dropbox, Google docs, Amazon S3, Microsoft OneDrive, and Github as well as attached to

communication mechanisms and applications like email, chat, and Slack. Today, just like local file systems, each of these storage solutions provides its own storage and naming mechanisms. And therein lies the problem: names were and are designed for users, but multiple disparate namespaces hurt the user experience. The user bears the burden of remembering personally irrelevant information: *where* they stored their documents. In the worst case, they must search through the individual storage silos. There is no practical/convenient way to quickly locate the item.

Existing solutions tie namespaces to storage silos, but this model does not fit with the way we use storage. We routinely access data from multiple silos, locating items by navigating or searching silo-by-silo.

We observe that names serve two purposes: names specify *location* and/or *context or semantic meaning* of the object. Context/meeting is most important to users; location is most important to the storage silo. Typically, storage systems fuse these naming purposes in such a way that it is not possible to relegate fixing the context/semantic naming problem to the Human Computer Interface (HCI) community.

The following (real world) scenario highlights some of the challenges.

A student from the country of Lemuria is destined for a summer internship in Camelot. Arranging for this internship requires many (many, many) different data objects: email messages with the host, offer letters, academic forms, a visa, boarding passes, project proposals, and more. How can our overwhelmed intern organize and/or find all the documents associated with their internship? We consider three approaches.

Meticulous: Place/copy every document into a single directory on their personal machine. Pros: It is simple to find everything. Cons: It requires conscious effort, pre-planning, and prescience. Everything is saved: emails, copies of physical documents, text messages, and project proposals. Our intern hopes they have correctly predicted what they will need in future. With more files and people involved, record keeping takes more time and people have different opinions on how to organize the files.

Haphazard: Leave everything where it is and search for it when you need it. Desktop search utilities such as Spotlight [125] and application-specific search tools make this possible, but this approach frequently requires searching across multiple silos, returns many more documents than intended, and misses those our intern [126].

Nirvana: Our tool on the intern's local machine creates a *personal namespace*. It combines conventional file system metadata with new (optional) user-provided metadata, relationships between items (e.g., an email message and the document attached to it, email messages with largely identical

contents, and documents shared among the same sets of people) and provides a navigation interface that uses this metadata to create collections of semantically related objects [115].

We tried to implement Nirvana outside the storage system [127]. We developed a visualizer that constructed personal namespaces by extracting existing and new properties (primarily relationships similar to those listed above) from multiple distinct data storage silos. We were surprised at the effectiveness of this approach to find related files, even across storage silos, particularly when combined with additional metadata that we generated.

Building Nirvana within the confines of existing storage silos has limitations: (1) using static file system attributes hindered us from providing useful ways to search and navigate data, (2) extracting file relationships across storage silos without explicit support for relationships as *attributes* was tedious, (3) our visualizer had no mechanism of maintaining privacy and security when storing these attributes outside the storage silos. Our observation is that dynamically generated, per-user namespaces show promise when used with today's multi-silo reality. However, this promise cannot be realized without storage system support and integration. We propose the Nirvana architecture, which separates *location names* from *context/semantic names*, and provides the latter *outside* of the storage system via a separate namespace service.

The idea of having distinct namespace providers delivering a global namespace is not new [34], [128], nor is the idea of using file metadata to generate namespaces [115]. Nirvana is novel in that it combines these pre-existing ideas, introduces new approaches, and is specifically designed to support today's multi-silo world. Nirvana combines 1) support for multiple namespace providers, 2) an expansive view of storage meta-data, 3) embracing search as an essential component of naming, and 4) a separation of user-visible names from silo-local names.

The rest of this paper introduces Nirvana and explores what system support is required to realize this vision.

A.4.3 Background

We are certainly not the first to propose either that naming is important [129], the need for better file system search [115], [116], [130], or personal namespaces [131]. The HCI community has similarly been observing both issues and potential alternatives for decades and it remains an open area of active research [126], [132]–[134]. We discuss existing technologies to determine those that we can use in constructing our solution.

Separation of location and semantic naming Cloud storage systems, such as Google Cloud and AWS S3 already decouple location names from human-readable names. In both systems, a user assigns to an object a *key name* and places the object in a *bucket* [135], [136]. The bucket hierarchy is flat, and a richer namespace is available for user convenience outside the object store itself. For example, S3 allows the user to simulate logical hierarchy using key name prefixes and delimiters, but the support for inferring this hierarchy is part of the AWS S3 console, not the S3 storage itself. Likewise, the S3 console supports a concept of folders. Google cloud provides similar features [137]. Neither objects nor buckets can be explicitly renamed within the storage system, but an entity *external* to it, Google Cloud Console, will simulate the renaming for users' convenience.

In each of these storage silos, something builds a namespace on top of them for us to access, but each of these constructed namespaces is *tied to the particular storage silo*. We do not know of a convenient method to navigate data using a common namespace across silos.

Application-defined namespaces As with naming, there are different kinds of namespaces in use today.

First, there are virtual machines or containers that provide applications, and even entire operating systems, with their own namespace for resource management. Within their isolated environment they are at liberty to organize objects as they please while the namespace provided by the hypervisor or the OS maps the VM/container-local name onto the backing store on disk.

Secondly, there are many application defined and managed namespaces. Examples include groupwares like Google Workspace or Microsoft Outlook managing contacts, emails, calendar events, notes, etc., all of which may contain attachments. Similarly, multimedia services store, organize and index videos and pictures providing a rich media library interface to the user [138]. Customer Relationship Management (CRM) systems store information about customers, orders, employees including documents for regulatory aspects. Our final example, wikis, hold any kind of information organized in pages and links between them. Each of these applications defines relevant metadata and decides how to store it. Even when it is possible to access the underlying assets, often useful information is lost in the process.

Third, users often resort to embedding metadata in the only place they have/know: the file and path name [3], [15]. There are limitations to this approach: file and path names are constrained in length, character sets are restricted, and as names grow longer they are less useful to humans.

For example, photographs from a camera or smart phone are devoid of anything but metadata, which means humans typically rely on thumbnail images to find things. In addition, these names with embedded metadata are inherently prone to conflicts: changing an attribute of the file can mean changing its name, thereby breaking connections between the object and other objects or programs that may have referenced it.

Search While conventional search (on a desktop or a mobile device) allows combining results of a keyword search across different applications and the web, it does not allow for discovery of objects linked by relationships more complex than sharing a keyword: e.g., lineage, time context (e.g., viewing one document while simultaneously writing a second document), etc. Furthermore, each search engine operates on whatever object metadata *it* can extract (and deems useful), but it could be faster and more effective if it could operate on explicit collections of metadata attributes securely and persistently maintained across storage silos.

Combining many namespaces Federated namespaces are an approach to bridging the gap between storage silos by binding two or more naming contexts [139]–[143]. The simplest example is file system mount points, another is CORBA (Common Object Request Broker Architecture), and its CosNaming service [144], [145]. A crucial limitation of federated namespaces is the assumption that bound namespaces must use similar ways to name and navigate objects. For example, on Unix you can mount another hierarchical file system, but not a key-value store. With CORBA, you can bind different CosNaming contexts, but not, say, an LDAP name server implementation. Furthermore, the federated model assumes that a storage system provides a human-readable namespace. We propose to separate the object storage from namespaces (Section A.5). This model makes adoption and integration of existing storage silos easier.

Extended Attributes Extended attributes have been supported by file system for more than 30 years, first appearing in the Novell Network File System in 1986. Since then many file systems have supported them, including HFS+ (Apple), UFS2 (UNIX), NTFS (Windows), and ext2 (Linux). Today, extended attributes are a common file system feature. Some systems support multiple extended attribute namespaces that separate user-defined attributes from system or security attributes, which require different permission levels to access [146]. POSIX v1.e initially had support for extended attributes as a

part of access control lists (ACL) but dropped it due to lack of interest[147]. OpenBSD dropped their support for the same reason[148]. The result is that we cannot rely on the file system to provide a standard and reliable way of saving user-defined metadata with a file and thus are not a viable solution for our problem. A list of issues that are caused by the current state of extended attribute support:

Space Limitations: Extended attributes are limited in size, e.g., ext4 allows up to 4KB, Amazon S3 objects support up to 2KB. The extended attribute space must be shared among multiple applications, so neither an application nor a user can know whether it is even possible to add a new attribute or what will happen if a security application does not have enough room left for its attributes.

Unique names: Since extended attribute space is shared by different applications, there is a potential for a clash in the names of the attributes from different applications. An extended-attribute key that works on one particular silo might not work on a different silo.

Loss of information: Not all underlying file systems support extended attributes. Therefore, we cannot save metadata on all file systems and cannot preserve metadata while moving a file from one file system to another.

Uncertified attribute modification: There is no method to prevent an application from accidentally or otherwise modifying an attribute that it does not own. Furthermore, since modifications are uncertified, it is impossible to determine the origin of the modifications and trust the values of the attributes.

Non-compatibility: Support and conventions are not consistent across implementations. Some file system attributes limit the length of attribute names; others limit the usable character set.

Ideally, all file systems would have a function to allow storing user-defined metadata. It should provide a mechanism to define multiple unique attribute namespaces to avoid collisions. It should also secure the namespaces so that application access can be restricted, e.g., only authenticated applications can write attributes in that namespace. Finally, it should allow users to define their trust level with distinct creators of attributes within that namespace. When we construct our global namespace, we can utilize such existing metadata, combining it with our own metadata, and storing it, possibly outside the storage silo containing the specific object.

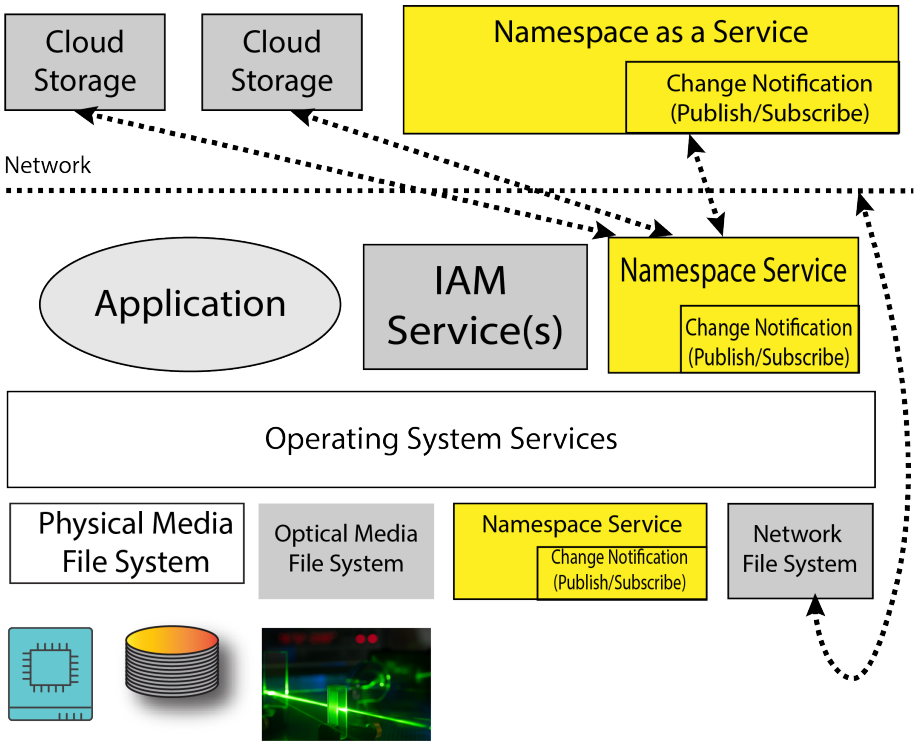

Figure A.2: Nirvana Systems Architecture.

A.4.4 Architecture

A.5 The Nirvana Architecture

We propose separating context/semantic naming from location and exposing this to users. This separation will enable users to navigate/search data across multiple silos independent of where data is stored, and their organization preferences. To realize this, we present the Nirvana architecture. In Nirvana, storage silos need not change how they store objects, assign them silo-local *location* names, support their choice of attributes, and provide ways for users and other services to access them. More interestingly, we absolve storage silos from providing *context/semantic* names. Instead, we introduce *Nirvana namespaces* and *namespace services*, which integrate one or more silos to provide *context/semantic* names and access capabilities. In the following discussion, we use the term *object* to refer to a discrete unit of storage, such as a file stored in an existing storage silo. Objects can reside on device-local

file systems or in the cloud. Users access objects via requests to namespace services, which can be either local or cloud-based (see Fig. A.2).

Namespaces In a Nirvana namespace, the name of a data object simply becomes a *collection of metadata attributes*. Attributes are tuples consisting of, e.g., type, name, value, and author. To date, we have identified three attribute types: *authoritative*, *annotated*, and *autogenerated*. Authoritative attributes are statements of fact provided by the silo, e.g., file size, creation time, content hash. The author of an authoritative attribute is the silo containing the object; if the object is replicated in multiple silos, the author is the originator of the object. Annotated attributes are provided by users or applications, e.g., Word document attributes, user-supplied tags, image metadata. The author of such attributes is the user or program providing the annotations. Finally, autogenerated attributes are produced by services, e.g., the cat video detector, a keyword extraction service. Authors are responsible for defining the semantic meaning of their attributes. Nirvana allows an unbounded number of attributes per object and requires only one: the authoritative location attribute, which is a set of identifiers that refer to the silo and silo-local name(s) by which an object can be accessed.

Namespace Services The namespace service provides support for three basic operations: (1) the ability to create a Nirvana reference, which is a set of metadata attributes; (2) the ability to map a reference to the underlying storage location and vice versa; and (3) the ability to query the namespace to find references that match a pattern. These references are immutable: deletion is achieved using a tombstone and update by creating a new reference, with a new creating timestamp. This preserves history and allows versioning, assuming storage silo support.

A *namespace provider* may also provide an API used by applications to interact with namespaces, allowing construction of new visualization and search tools. For example, if you are searching from your smartphone, you can find a specific document, regardless of where it is stored, whether that is your cloud storage system, a desktop computer, or some other storage location and even if it is not presently accessible. Conversely, this model can be constructed to permit maintaining multiple distinct namespaces, so that each user is able to see *views* of their own data.

Locating specific objects From our description so far, it seems that Nirvana is all about providing fertile ground for effective search, but it may be unclear

how the user could *locate* the specific object they need. We provide an example to illustrate how Nirvana can support this.

In traditional systems, a user gives a file a memorable name, e.g., `eddie.txt`. In addition, hierarchical directory structure provides additional context for the object, helping the user locate the precise instance of `eddie.txt` that they need. For example, there can be a `faculty/recommendations/eddie.txt` and `papers/fantasstic/eddie.txt`. Nirvana supports locating objects via memorable names by assigning the object a user-provided attribute, e.g., `eddie.txt`. Similarly, to simulate navigation by directory hierarchy, an object can be assigned attributes corresponding to the path components. In this way, Nirvana can also support legacy tools relying on hierarchical directory namespaces.

At the same time, Nirvana can help users go beyond using memorable names. For example, in the experimental science community, users traditionally embed the experiment context into the name of their data files [3]. In Nirvana, users can explicitly store those metadata elements in a namespace. Then, grepping for, say, `experiment-data.txt` can return multiple instances, but their metadata attributes can be explained via `stat`, and a ``` ``diffstat'' ``` command could also show the differences in the attributes between the two references.

While hierarchical file system provide us with path and file names for search, our approach can make use of other augmented metadata elements, allowing us to narrow the results.

Sharing, Access Control and Privacy Enabling safe sharing requires a clear model of access control and a clear understanding of possible privacy implications. While traditional file systems do have a vague notion of namespace access control using directory permissions, we are now faced with two explicit levels requiring access control: the object itself and the metadata.

What rights exist on each level and how do they relate to each other? In Nirvana, possession of the reference does not imply any rights to access the object, since access is managed by the storage silo itself. The namespace provider has the ability to encode encrypted values within metadata elements, which also allows storing of sensitive information within the name without compromising security — anyone with the Nirvana REF must still also have the relevant key to understand the value of specific attributes. Similarly, this permits storing metadata in a third party service without compromising the contents of the metadata attribute.

An important element of the Nirvana namespace model is the ability to

identify not only the *type* of a metadata attribute, but also the *authority* that provided such data. To accomplish this, we need to have a mechanism for allowing verifiable identification for the metadata attributes that make up the name. However, rather than propose a new identity and access management (IAM) service, we simply note that we expect Nirvana to work with a least one such service (Fig. A.2). In addition, this model also allows for a specific treatment of trust relationships that need not rely upon the namespace itself. In essence, this allows the user to define their own "security rings of trust."

Nirvana Namespaces detect storage changes via a publish/subscribe model, similar to prior work [15], [149].

A.5.1 Research Opportunities

File System Evolution With a distinct namespace implementation, we need to consider questions on how to optimize their interaction. For example, should we permit the namespace implementation to store its metadata within the storage silo? How can the storage silo access metadata from the namespace? How can we optimize storage silos when they are not burdened with the different needs of namespace and storage management? Can we provide mechanisms to notify the namespace when changes occur within the storage silo? These questions arise because we separate the traditional, hierarchical file system structure we have known for more than a half century.

Traditionally, users see storage as a convolution of location and its context within the storage, i.e. the path of the file defining its location and context. Performing a file system operation was reflected on both the file system context and the backing store at the same time. A clean separation of this two concepts into the storage (location name) and namespace (semantic/context name) raises fundamental questions on what kind of operations are supported by each of those two concepts, how they relate to each other, and how closely coupled they are.

Differential Privacy and Security The notion of access control mentioned above raises the question of how the security model used in this two layer system, as well as potential privacy issues: the meta-data may reveal much information about the object itself, even its presence, replication factor or temporary unavailability within a namespace is leaking information. How do we control this kind of information flow on the namespaces and the backing

store?

Can we provide some form of differential privacy to the user of the name space? Effectively by prohibiting the user from enumerating the content of the namespace, and restricting queries to those who only produce one result – in the sense you need to know precisely what you are looking for. As a more general aspect: how do we effectively prevent data leakage?

For example, in current namespaces there is no simple mechanism for finding all of the references to a specific object (e.g., the embarrassing photo that needs to be forgotten). Nirvana provides the ability to find such references. Can we create a GDPR compliance system that guarantees to expunge such references?

Using Metadata to Optimize Storage Currently, we have to explicitly decide on where we want to store our data by manually selecting from various data silos providing different properties, guarantees and methods of sharing. Having a rich namespace containing meta-data for the objects, could we leverage this information to deduce the best method of storage? For instance, the origin of the information may influence the selection of the data silo, whether data is to be encrypted, an expiration date assigned, creating multiple instances of the object in different silos, or whether we may want to keep track of different versions of the object. Moreover, one may want to keep multiple, synchronized copies of the same object to support different access pattern or ensure locality. How does regulation come into play (e.g., a object must not leave a certain jurisdiction)? One of the central aspects of storage optimization is the question of which entity performs this? This will likely require some form of synchronization among multiple namespaces that know about the object to ensure stability and avoid frequent changes to the storage.

A.5.2 Conclusion

We demonstrated providing useful ways to navigate/search across storage silos is not simply an HCI problem: it requires new system support. We proposed Nirvana that, at its heart, separates location names from context/semantic names, and maintains those names in a cross-silo name service separate but complementary from the storage. We presented research questions to address when building Nirvana.

A.6 HotStorage 2021

A.6.1 abstract

Users store data in multiple storage silos, such as Google drive, Slack, email, Dropbox, and local file systems that mostly rely on traditional user-assigned names. A user who wants to locate a document that she saved while having a conversation with her colleague on a specific subject last month will have a hard time finding that document if she doesn't remember in which silo it was stored or what name it was given.

Prior work that introduced a *Placeless* storage architecture enabled cross-silo search using semantically meaningful attributes, while other prior work used data provenance to construct a user's *activity context* (e.g., what they were doing at the time they created or accessed data) to aid in document location. We take a position that despite these prior systems that demonstrated rich semantic search capabilities, we still cannot provide these capabilities using existing system APIs and abstractions.

We explain that this is not simply an HCI problem and identify the systems problems that must be solved to realize this vision. We present *Kwishut*, an architectural blueprint for enabling semantically rich, multi-silo data management.

A.6.2 Introduction

Today's file systems provide two primary functions: a way to store chunks of data and names that provide users with the familiar metaphor of a file cabinet (i.e., folders and documents). Much has changed since we adopted this design.

Now most data resides not on local file systems but on myriad services such as Dropbox, Google docs, Amazon S3, Microsoft OneDrive, and Github, as well as attached to communication mechanisms and applications such as email, chat, and collaborative communication platforms (e.g., Slack, Discord).

Today, just like local file systems, each of these storage solutions provides its own storage and naming mechanisms. Pity the user Alice who wants to find the file that was sent to her by Bob while they were having a slack conversation about cool papers in HotStorage 2020, if she remembers neither the name of the file nor whether she stored it in Dropbox, Google drive or her local file system.

The *Placeless* architecture [21] provided an elegant solution to this problem by enabling search across different storage silos using semantically meaningful

names. Each file was annotated with a rich set of attributes, determined by the user or generated by software, and a naming service, spanning silos, searched the entire collection of user files using these semantically meaningful attributes. *Placeless* was a huge improvement over isolated storage silos and semantically-poor names, but it did not solve Alice's problem. Finding Alice's document requires that we track files across storage silos and *activity contexts*. An activity context describes the other activities that were happening at the same time a document was accessed. This context might include applications, such as Slack, email or a web browser, which are not file systems in any traditional sense, but can be sources and destinations for data and for its semantically meaningful context.

The only solution of which we are aware that captures activity context is Burrito [3], which used data provenance to keep track of a user's activity context, i.e., the applications they were running and actions they were taking while examining a particular file. Unfortunately, Burrito is a desktop application that was intended neither to work across multiple devices nor to span multiple, remote storage silos. While it introduced the idea of activity context sensitive search, it did not address any of the semantic searching issues of Placeless and it did not consider the privacy consequences of storing user context in a distributed environment.

We posit that **1) user naming should be entirely decoupled from local naming and 2) users need customizable and personal namespaces.** We present *Kwishut*[1], an architecture embodying this position, leveraging existing infrastructure where possible and extending it where necessary. *Kwishut* uses separate metadata and naming services coupled with user activity monitors. *Kwishut* is designed to allow incorporation of existing storage, metadata, and naming services without modification, while providing enhanced functionality when services support *Kwishut* features. We limit discussion to the systems infrastructure required to realize this vision; current storage management interfaces (e.g., file browser) can use *Kwishut* namespaces directly, while the availability of rich metadata in *Kwishut* enables HCI research on better ways for users to identify and find their data.

We begin with use cases motivating the need for *Kwishut* (§A.6.3), highlighting specific features missing from today's storage and naming services. Next, we present the *Kwishut* architecture (§A.6.5) and future research directions it enables (§A.6.8). We then discuss how *Kwishut* builds upon prior work (§A.6.9) and conclude summarizing our position (§A.6.10).

[1] *Kwishut* means "naming" in a native North American language.

A.6.3 Why we need *Kwishut*

Feature	Existing Technologies	No Solution
ACTIVITY CONTEXT	timestamps and geo-location, image recognition, browsing history, ticketing systems, application-specific solutions like Burrito [3].	Link related activity across apps, record browsing history and chat conversations relevant to the creation of the data object, storing it in ways that are secure and compact.
CROSS-SILO SEARCH	Search by name, creator, content across silos, app-specific searches (e.g., Spotlight)	Unified search across all kinds of storage, including file systems, object stores, apps and devices
DATA RELATIONSHIPS	De-duplication of documents, versioning of specific files, git ancestor relation	Explicit notion of data identity, tracking different versions across different silos as data is transformed
NOTIFICATIONS	File watchers (INotify), synchronization status, manually inspecting modified time	Ability to subscribe to specific changes on attributes
PERSONALIZED NAMESPACE	Hierarchy plus hard/soft links. Use of tags.	Creating personalized namespaces with with flexible data organization and views

Table A.4: Use-case driven functional requirements.

We identified five categories of information that are necessary to facilitate the integration of semantically meaningful naming with user activity context. Unfortunately, to varying degrees, these features cannot be provided by today's storage system architecture. We introduce these categories, summarize them in table Table A.4, and then present use cases demonstrating how they facilitate data management.

Feature Wish List

Activity Context: As Burrito demonstrated [3], the *context* in which data were accessed or created is often a useful attribute on which users wish

to search, e.g., "*I'm looking for the document I was editing while emailing Aki about their favorite wines.*" To the best of our knowledge, there is no modern system that supports queries using rich context across applications. We might be able to use timestamps or application-specific tags or history information in queries, but it is laborious, if not impossible, to intersect data from multiple applications and/or multiple silos.

Cross-Silo Search: Users share documents in myriad ways: via messaging applications, on cloud storage services, and via online applications. Users should not need to remember which mechanism was used to share a particular document and should have some easy way of organizing and searching through a collection of such distributed documents.

Data Relationships: Documents can be related in arbitrary ways. This relationship information can be used to facilitate and enable better search results. So far, we have identified three specific relationships that are particularly important:

1) copy is a bit-for-bit identical replica of some data, in other words two items with different names store the same data.

2) conversion is a reversible, repeatable transformation that changes the representation of data, without changing its semantics, e.g., converting a CSV file into JSON.

3) derivation refers to data that has been computationally derived from another object by altering its content, e.g., adding a row to a spreadsheet.

While storage systems can recognize copies, they cannot distinguish conversions from derivations. However, from a user's perspective, these operations are quite different: a conversion can be repeated, which is not necessarily true of a derivation.

Notifications: Users frequently want to be notified when documents change, and many storage services offer this functionality. However, users might also want notification when data on which they directly or indirectly depend changes. This requires both a notification system and an awareness of the data relationship between different objects.

Personalized Namespaces: Users have different preferences and mental models to organize their documents, a source of conflict in a multi-user setting. We need a way to provide each user the ability to personalize their document structure.

A.6.4 Use Cases

The following use cases illustrate how the features described above arise in common place activities.

Data Processing: Aki and Fenix are preparing a report summarizing their work on a data analysis project for a customer. Fenix sends an email to Aki containing a CSV file with original data. Aki opens this document in Excel, formats and filters it, adds additional data from a corporate storage silo, and then returns the Excel document to Fenix on Slack. Fenix is away from their desk when it arrives, so they open it on their phone, uploading it to a cloud drive. Fenix then sends the link to Aki for editing with update notifications. Finally, Fenix sends a PDF of the report to the compliance officer who promptly asks, "Where did this data come from?"
Feature Analysis: This use case highlights the need for 1) DATA RELATIONSHIPS, as it has instances of copies, conversions and derivations, 2) CROSS-SILO SEARCH, as these items are located in multiple silos and accessed by multiple devices, and 3) NOTIFICATIONS, as update notifications need to be distributed to designated users.
Delete Request: Some time later, the compliance officer requests that all documents containing a customer's data must be deleted. To help with finding all relevant customer data, Dagon joins the project and examines the report and requests the original data from which it was produced. Aki remembers that they gave the original data to Fenix shortly after collecting it, but does not remember the name, location, or even how the relevant files were transmitted. Thus, Aki has to manually search possible locations and applications, sendsing references to documents to Dagon, who then starts organizing these files to methodically identify the ones that might contain the customer's data. In the process, many of the other team members' references to the documents stop working.
Analysis: This use case illustrates the need for 1) ACTIVITY CONTEXT to capture data that has been collected while interacting with the customer, 2) DATA RELATIONSHIPS to identify related documents, 3) CROSS-SILO SEARCH to easily locate relevant documents across data silos, and 4) PERSONALIZED NAMESPACE to create a individual data organization.

From Use Cases to Architecture

Each use case and feature class suggests capabilities that are unavailable today.

In Table A.4 we identify existing technology that can be brought to bear on the problem while teasing apart the precise details that are missing.

Repeatedly, we find that critical information necessary to provide a feature is unavailable, that providing such information is non-trivial, and that obtaining it creates a collection of privacy challenges.

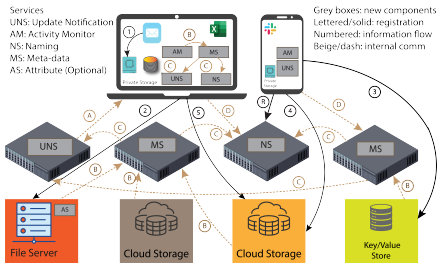

Figure A.3: *Kwishut* Architecture (§A.6.5).

A.6.5 Architecture

Kwishut is a family of services that enable sophisticated search and naming capabilities. The key features that differentiate *Kwishut* from prior work are: *1)* incorporating object relationships as first class meta-data, *2)* federating meta-data services, *3)* recording activity context, *4)* integrating storage from multiple silos, and *5)* enabling customizable naming services. Data continues to reside in existing and to-be-developed storage silos. *Kwishut* interacts with these silos, collects and captures metadata, and provides a federated network of metadata and naming services to meet the needs of the use cases in §A.6.3.

A.6.6 *Kwishut* Services

Figure A.3 illustrates the *Kwishut* architecture. *Kwishut* allows for different deployment scenarios. The five services can be run independently, they can be co-located and bundled together to run on a local device, integrated into an OS, or available as web-based services.

In the discussion below, parenthesized numbers and letters refer to the arrows in Figure A.3. There are five main components:

1) Metadata servers (MS) are responsible for storing attributes and provide a superset of capabilities found in existing metadata services [25], [39]. Users can register an MS with activity monitors or attribute services, which allows the MS to receive updated attributes from storage objects and activities (B). Thus, there can be multiple sources of attributes including the user itself. Metadata servers may retain the full or partial history of attribute updates or maintain only the most recent value.

2) Namespace servers (NS) connect to one or more MS and use the metadata to provide users with a personalized namespace that allows both manual organization (i.e., a hierarchical namespace) and rich search capabilities.

Users can register with an NS (R) that uses one or more MS to obtain relevant attributes from them (C). Additionally, users can be part of a corporate NS that allows sharing of their select metadata with other users via standard enterprise public-key cryptography.

3) Activity monitors (AM) run on the user's devices. Their main function is to observe temporal relations, activity context, and relationships between objects on a user's device and transmit them to an MS (D).

4) Attribute services (AS) extract attributes from storage objects and transmit them to an MS (B). An AS might be invoked on updates, run once or periodically. For example, a file system AS would update the object's metadata with basic attributes such as size or modification time. There can be many AS that extract more "interesting" attributes, e.g., image recognition, similarity, or other classifiers.

5) Update notification server (UNS) provides notification mechanisms. Users can register interest in changes of attributes or underlying storage and will receive a message on change events (A) to which they have access.

A.6.7 *Kwishut* working example

To make the *Kwishut* architecture concrete, we revisit our use-cases from §A.6.3 and walk through parts of it to illustrate how *Kwishut* supports the various actions and events.

Storing the e-mail attachment. Aki's act of saving the CSV file that Fenix sent in email corresponds to the creation of a new object on the file server silo, i.e., the file system (4). The file server is *Kwishut*-aware, so the AS co-located with it extracts attributes from the document and forwards them to the MS (B).

The AM on Aki's laptop detects that the CSV file came via company email from Fenix. It then captures the activity context identifying the relationship between the e-mail and the CSV file and transmits it as additional metadata about the CSV file to the MS (that already contains metadata extracted by the AS). Moreover, because there is a company-wide namespace service, *Kwishut* establishes that the e-mail attachment, the CSV in the file server, and the one on Fenix's laptop (from which the file was sent) are exact copies of each other.

Many applications already record some form of activity context, e.g., chat history, browsing history. Such histories provide a rich source of additional metadata. Other activity context, specifically the relationship between objects, such as the fact that a particular file was saved to a local storage device from an email message, requires more pervasive monitoring as found

in, e.g., whole provenance capture systems [46]. *Kwishut* is agnostic about the precise data that comprises activity context, but allows for storing and accessing activity context as metadata.

Creating the Excel file. Next, Aki opens the CSV file using Excel and stores it as a spread sheet. This creates a new object. The AM detects that the newly created spreadsheet is a conversion from the CSV file, either via a notification from *Kwishut*-aware Excel or by monitoring the system calls executed on the local system. Aki proceeds to modify the data by filtering it in Excel and saving the changes. The AM records this event and updates the meta-data of the spreadsheet to record the derivation-relationship. Ideally a *Kwishut*-aware version of Excel specifies to the AM the exact type of the relationship (in this case a derivation); otherwise the AM informs the MS about an unspecified data relationship by observing the opening of a CSV file and a subsequent creation of the Excel file.

Sharing the spreadsheet. As Fenix receives the Excel file from Aki via Slack on their phone, a sequence of metadata events similar to those described earlier takes place, except the phone does not run a local NS or MS. Fenix now uploads the file to the company's cloud drive (4). The MS (by way of AS) reflects the creation of a new object and records its remote location. The use of a company-wide namespace and metadata service enables *Kwishut* to record that the file in the cloud drive is, in fact, a copy of the one received via Slack. Further, Fenix informs their personal NS that they wish to notify Aki about all updates to the file on the cloud drive. Thus, whenever an AS sends updated attributes to the MS, Fenix receives a notification.

Data origin and delete requests. When the compliance officer asks about the origin of the data, Fenix can query the corporate NS to obtain the complete history of the report. This includes the spreadsheet from which the report was derived and the e-mail or Slack messages that transmitted the files. The corporate NS was configured to be aware of the locations of the collaborating users' personal NS. Moreover, because of the activity contexts captured by the AM, *Kwishut* is able to identify documents that were created during any activity involving the customer whose data must be deleted. Starting from these documents, and by using the relationship of documents, Dagon was able to find all relevant objects and delete them, including the e-mail and Slack messages.

Note that unlike existing systems, *Kwishut* is able to efficiently find related objects across storage silos. Operating systems already provide users with indexing services to accelerate search of local files. This search can be made cross-silo by mounting and enabling indexing on network shares (e.g., Windows Desktop Search), or by interfacing with specific applications such

as e-mail (e.g., MacOS Spotlight, or Android search). The problems with indexing on a large network storage repository are resource limitations such as bandwidth and local storage that may render the system unusable during indexing. In contrast, *Kwishut* addresses these limitations by delegating indexing and storage to one or more services.

NS are responsible for providing efficient search functionality. *Kwishut* uses AS to keep attributes up to date with object modifications. Lastly, *Kwishut* supports coordinated search among one or more local and remote NS, allowing, for example, a user to search across both their local NS as well as their employer's NS.

A.6.8 Future Directions

We now explore a few research directions that *Kwishut* suggests.

Attribute Security and Integrity. *Kwishut* decouples naming and attributes from the storage object. This opens up a research direction on the security model of attributes themselves. Are the permissions on the attributes similar to the ones on the storage objects themselves? Can a user change an attribute in its local namespace, but not in the company wide one? This segues into the question of attribute integrity/quality: not all attribute sources have the same trust-level. For instance, a user might label an image "dog," while the image recognition AS might label it "cat".

Privacy. *Kwishut* collects a lot of metadata across multiple communicating channels and storage silos, including activity data. This raises the question of how to manage these metadata in a privacy-preserving manner.

Interface Design. We presented a system architecture that provides a rich context to search and organize storage objects. We envision that this will provide the foundation for new directions in HCI research: By using individual namespaces, we can dynamically organize and visualize documents and other storage objects, and seamlessly navigate and locate related documents providing a new user experience.

Relationship-based Queries. *Kwishut* tracks relationships between storage objects. These relationships provide minimal data provenance [45] allowin users to locate the chain of related documents originating in a specific activity context.

These relationships are most naturally expressed as graphs, where nodes are objects, and edges are the relationships between objects. Edges could have weighted-labels, indicating the type and importance of their relationship.

This enables more sophisticated data-analysis beyond pure content-based indexing by using graph queries.

For instance, lineage queries (i.e., tracing the history of an object) are path traversals, which are challenging to implement efficiently in conventional storage systems. This suggests that the NS and/or MS require sophisticated storage and query mechanisms.

A.6.9 Related Work

Kwishut draws on prior work in semantic file systems, using search to locate documents, and federated naming systems.

Semantic File Systems.

Although we introduced our desire for semantically meaningful names with reference to the Placeless architecture, the idea originated in the systems community with the semantic file system [22], SFS.

SFS used automatically extracted attributes to construct virtual directories that contained collections of semantically related documents.

There exist many extensions or variants on this theme such as per-process namespaces [23], inverting the database/file system layering to build file systems on top of queriable databases [24], manually tagged file systems [26], constructing semantic metadata stores from distributed storage [25], and systems that manage conventional and semantic structures in parallel [27]. *Kwishut* represents another step in this evolution. It extends prior work by combining semantic naming with user activity context and is designed for today's multi-silo'd storage encompassing everything from mobile devices to desktops to object stores to cloud storage.

Search. An alternative to creating a semantically meaningful name space is to enable extensive metadata-based search. Desktop tools such as Apple's Spotlight, Linux KDE Baloo, and Windows Desktop Search adopt this approach. However, breakthroughs in web search (i.e., incorporation of pagerank [31]) demonstrated that the relationship among objects is at least as important as the metadata itself. The efficacy of provenance-assisted search [28]–[30] demonstrates that history, in addition to relationships enhance users' ability to locate documents. However, searching for documents is fundamentally different from naming. Search-based approaches rely either on a user to select the correct item from many presented or on the sufficiency of providing *any* relevant document. However, naming requires the ability to identify a specific document. *Kwishut* is designed to support both searching and uniquely identifying a specific document.

Multi-silo Data Aggregation. UNIX mount points [37] are perhaps the first instance of federating namespaces. Distributed federation, as provided by distributed file systems such as NFS [32] and AFS [34] followed soon

after adoption of local area networks. With the advent of cloud storage, there has been work in federated namespaces that span cloud stores [38], [39]. Nextcloud (https://nextcloud.com) allows users to connect multiple Nextcloud instances and integrate with FTP, CIFS, NFS and Object stores. Yet, documents are still organized in a classic hierarchical structure. Peer-to-peer sharing networks (e.g., IPFS [41]) implement a distributed file system where nodes advertise their files to users. MetaStorage [33] implements a highly available, distributed hash table, but with its data replicated and distributed across different cloud providers. Farsite [36] organizes multiple machines into virtual file servers, each of which acts as the root of a distributed file system. Comet describes a cloud oriented federated metadata service [39].

A.6.10 Conclusion

We have presented our position that we need *Kwishut*, a storage architecture that decouples naming from the storage location of documents and data objects, provides customizable and personalized namespaces, and that makes relationships between documents a first class citizen. With *Kwishut*, users will be able to organize, share and find their data conveniently across multiple storage silos using a rich set of attributes breaking away from the rigid, hierarchical organization.

We expect *Kwishut* will enable a broad area of research in HCI exploring new ways to visualize and interact with data using the mechanism's provided by *Kwishut*. Moreover, we expect *Kwishut* to provide interesting scenarios for security and privacy research in storage systems.