# Position: Naming is Hard

Anonymous Author(s)

## 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.

## 1 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 [1] 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 [2], 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* (§2), highlighting specific features missing from today's storage and naming services. Next, we present the *Kwishut* architecture (§3) and future research directions it enables (§4). We then discuss how *Kwishut* builds upon prior work (§5) and conclude summarizing our position (§6).

## 2 WHY WE NEED *KWISHUT*

We identified five categories of information that are necessary to facilitate the integration of semantically meaningful

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

| Feature | Existing Technologies | No Solution |
|---|---|---|
| ACTIVITY CONTEXT | timestamps and geo-location, image recognition, browsing history, ticketing systems, application-specific solutions like Burrito [2]. | 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 1: Use-case driven functional requirements.**

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 1, and then present use cases demonstrating how they facilitate data management.

## 2.1 Feature Wish List

**Activity Context:** As Burrito demonstrated [2], 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 Addison 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.

## 2.2 Use Cases

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

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

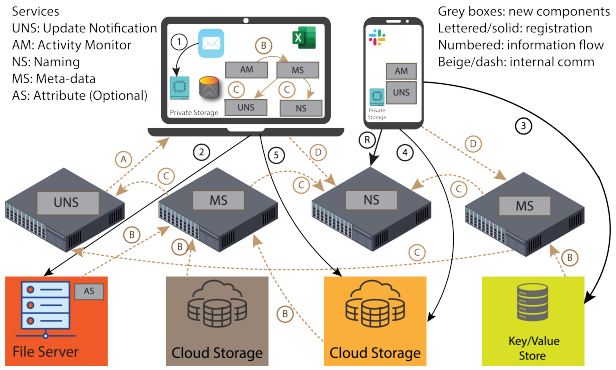


**Figure 1: *Kwishut* Architecture (§3).**

*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, Bailey joins the project and examines the report and requests the original data from which it was produced. Addison remembers that they gave the original data to Cameron shortly after collecting it, but does not remember the name, location, or even how the relevant files were transmitted. Thus, Addison has to manually search possible locations and applications, sendsing references to documents to Bailey, 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.

## 2.3 From Use Cases to Architecture

Each use case and feature class suggests capabilities that are unavailable today. In Table 1 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.

# 3 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 §2.

## 3.1 *Kwishut* Services

Figure 1 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 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 [3, 4]. 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.

### 3.2 *Kwishut* working example

To make the *Kwishut* architecture concrete, we revisit our use-cases from §2 and walk through parts of it to illustrate how *Kwishut* supports the various actions and events.

**Storing the e-mail attachment.** Addison's act of saving the CSV file that Cameron 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 Addison's laptop detects that the CSV file came via company email from Cameron. 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 Cameron'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 [5]. *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, Addison 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. Addison 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 Cameron receives the Excel file from Addison 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. Cameron 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, Cameron informs their personal NS that they wish to notify Addison about all updates to the file on the cloud drive. Thus, whenever an AS sends updated attributes to the MS, Cameron receives a notification.

**Data origin and delete requests.** When the compliance officer asks about the origin of the data, Cameron 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, Bailey 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.

## 4 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 [6] 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.

## 5 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 [7], 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 [8], inverting the database/file system layering to build file systems on top of queriable databases [9], manually tagged file systems [10], constructing semantic metadata stores from distributed storage [4], and systems that manage conventional and semantic structures in parallel [11]. *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 [12]) demonstrated that the relationship among objects is at least as important as the metadata itself. The efficacy of provenance-assisted search [13–15] 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 [16] are perhaps the first instance of federating namespaces. Distributed federation, as provided by distributed file systems such as NFS [17] and AFS [18] 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 [3, 19]. 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 [20]) implement a distributed file system where nodes advertise their files to users. MetaStorage [21] implements a highly available, distributed hash table, but with its data replicated and distributed across different cloud providers. Farsite [22] 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 [3].

## 6 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.

# REFERENCES


[1] Paul Dourish, W. Keith Edwards, Anthony LaMarca, John Lamping, Karin Petersen, Michael Salisbury, Douglas B. Terry, and James Thornton. Extending document management systems with user-specific active properties. *ACM Trans. Inf. Syst.*, 18(2):140–170, April 2000. ISSN 1046-8188. doi: 10.1145/348751.348758. URL https://doi.org/10.1145/348751.348758.

[2] Philip J. Guo and Margo Seltzer. Burrito: wrapping your lab notebook in computational infrastructure. In *TaPP'12 Proceedings of the 4th USENIX conference on Theory and Practice of Provenance*, pages 7–7, 2012.

[3] Cong Wang, Komal Thareja, Michael Stealey, Paul Ruth, and Ilya Baldin. Comet: A distributed metadata service for federated cloud infrastructures. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2019. doi: 10.1109/HPEC.2019.8916536.

[4] Yu Hua, Hong Jiang, Yifeng Zhu, Dan Feng, and Lei 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*, pages 1–12, 2009. doi: 10.1145/1654059.1654070.

[5] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. Practical whole-system provenance capture. In *Symposium on Cloud Computing (SoCC'17)*. ACM, 2017.

[6] Lucian Carata, Sherif Akoush, Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, Margo Seltzer, and Andy Hopper. A primer on provenance. *Commun. ACM*, 57(5):52–60, May 2014. ISSN 0001-0782. doi: 10.1145/2596628. URL https://doi.org/10.1145/2596628.

[7] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole. Semantic file systems. In *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, SOSP '91, page 16–25, New York, NY, USA, 1991. Association for Computing Machinery. ISBN 0897914473. doi: 10.1145/121132.121138. URL https://doi.org/10.1145/121132.121138.

[8] Rob Pike, Dave Presotto, Ken Thompson, Howard Trickey, and Phil Winterbottom. The use of name spaces in plan 9. *SIGOPS Oper. Syst. Rev.*, 27(2):72–76, April 1993. ISSN 0163-5980. doi: 10.1145/155848.155861. URL https://doi.org/10.1145/155848.155861.

[9] Michael A. Olson. The design and implementation of the inversion file system. In *USENIX Winter 1993 Conference (USENIX Winter 1993 Conference)*, San Diego, CA, January 1993. USENIX Association. URL https://www.usenix.org/conference/usenix-winter-1993-conference/design-and-implementation-inversion-file-system.

[10] Stephan Bloehdorn, Olaf Görlitz, Simon Schenk, and Max 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.

[11] Daniele Di Sarli and Filippo Geraci. Gfs: A graph-based file system enhanced with semantic features. In *Proceedings of the 2017 International Conference on Information System and Data Mining*, ICISDM '17, page 51–55, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348331. doi: 10.1145/3077584.3077591. URL https://doi.org/10.1145/3077584.3077591.

[12] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[13] Sam Shah, Craig A. N. Soules, Gregory R. Ganger, and Brian D. Noble. Using provenance to aid in personal file search. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ATC'07, USA, 2007. USENIX Association. ISBN 9998888776.

[14] Annajiat Alim Rasel and Mohammed Eunus Ali. Uprove2: privacy-aware, scalable, ubiquitous provenance to enhance file search. In *2016 International Conference on Networking Systems and Security (NSysS)*, pages 1–5. IEEE, 2016.

[15] Jinjun Liu, Dan Feng, Yu Hua, Bin Peng, Pengfei Zuo, and Yuanyuan 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*, pages 597–611. Springer, 2015.

[16] Dennis M. Ritchie and Ken Thompson. The unix time-sharing system. *Commun. ACM*, 17(7):365–375, July 1974. ISSN 0001-0782. doi: 10.1145/361011.361061. URL https://doi.org/10.1145/361011.361061.

[17] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the sun network filesystem. In *Proceedings of the Summer USENIX conference*, pages 119–130, 1985.

[18] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, 1988.

[19] Alysson Neves Bessani, Ricardo Mendes, Tiago Oliveira, Nuno Ferreira Neves, Miguel Correia, Marcelo Pasin, and Paulo Verissimo. Scfs: A shared cloud-backed file system. In *USENIX Annual Technical Conference*, pages 169–180. Citeseer, 2014.

[20] Juan Benet. Ipfs - content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[21] David Bermbach, Markus Klems, Stefan Tai, and Michael Menzel. Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 452–459, 2011. doi: 10.1109/CLOUD.2011.62.

[22] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. *SIGOPS Oper. Syst. Rev.*, 36(SI):1–14, December 2003. ISSN 0163-5980. doi: 10.1145/844128.844130. URL https://doi.org/10.1145/844128.844130.