Not Dead Yet

Hierarchical File Systems Won't Die

Tony Mason
The University of British Columbia
fsgeek@cs.ubc.ca

Margo Seltzer
The University of British Columbia
mseltzer@cs.ubc.ca

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.

1 INTRODUCTION

The hierarchical model is deeply embedded in our fundamental precepts of data organization. Introduced in Multics [16, 18], the authors describe the hierarchical file system, but they also introduce *links*, suggesting an imperfect model.

Almost all systems that followed Multics adopted the hierarchical namespace, e.g., AT&T [75] and Digital [86]. Early UNIX work suggests limitations in the model; they describe the importance of the *permuted index server*, a program that constructed concordance databases of file contents based upon keyword identification [75].

File systems, as implemented today consist of two separate components: the *data storage* and the *name space* [5, 6, 67, 80]. Data storage has evolved; gone are the days when everything a programmer needed could be stored on a single magnetic tape or named by the numbers 1 to 2048 [47, 98]. In 1965, 1TB of data required 8.4 tons of hardware; in 2019, less than 50 grams of media and hardware are needed. In 2025, IDC estimates that there will be 163 zettabytes (ZB) of data [74]. The namespace, however, remains fundamentally unchanged from its conceptual beginnings in Multics; the namespace need not be tightly coupled to the underlying storage, although it frequently is [12, 17, 60].

The world wide web has shifted user expectations to a search-based model [76]. Google revolutionized search on the internet by leveraging metrics that exploit the inherent *graph-structured* nature of the web. Typical users think of neither local data nor web data as hierarchical. Some systems researchers may still employ the hierarchical mental model, but then use "sophisticated" tools, such as `find` and `grep`,, to locate relevant information in their own files — further evidence the file systems namespace is insufficient.

This is a *systems* problem — not *merely* a Human Computer Interaction (HCI) problem — because the namespace belongs to the system. The system already maintains a limited set of relationships for files, we claim that relationships are the foundation of a better namespace model. When starting with relationships as first class objects, the system level design of the name space changes dramatically.

Thus, we propose a file system that natively supports relationships as a fundamental feature. Note: the question is not *should* we be managing relationships — we already do — but what additional relationships we should manage. The pure hierarchical structure is a *tree*, which captures one primary relationship; when we support an arbitrary number of equally important relationships, we move from a *tree* to a *graph*. This creates questions we consider: 1) How do we best capture, store, query, and manage such relationships, and 2) How do we best present these relationships to users. The first is a systems problem and the subject of the rest of this paper. The second is an HCI problem — and one the HCI community shows considerable interest in answering [14, 29, 30, 32, 34, 36–38, 40, 46, 54, 89, 91–94].

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.

2.1 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 (§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.

2.2 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 [10]. 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 [68] and other index-based structures [81] 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.

2.3 Graph FS Model

We set out a basic description of our core objects in Table 1 and a demonstrative set of example relationships in Table 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.

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

Table 1: Graph File Systems Terminology

Every file has a unique identifier, such as a **UUID**, similar to an inode number or object ID. We do not rely upon *names* as they are simply mutable properties.

Relationship	Description
similar	Similarity measure, e.g., [59]
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 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

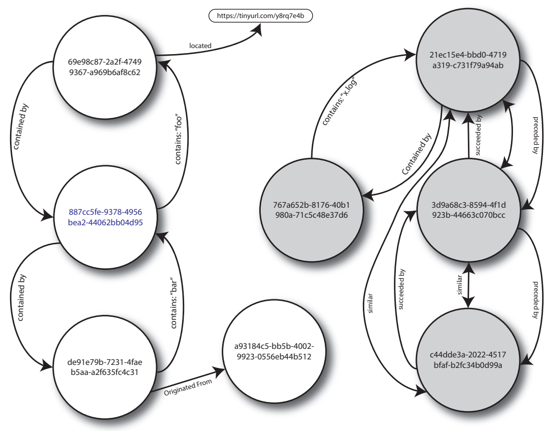
https://tinyurl.com/y8rq7e4b
located
69e98c87-2a2f-4749 9367-a969b6af8c62
contained by
contains: "foo"
887cc5fe-9378-4956 bea2-44062bb04d95
contained by
contains: "bar"
de91e79b-7231-4fae b5aa-a2f635fc4c31
Originated From
a93184c5-bb5b-4002-9923-0556eb44b512
contains: "x.log"
21ec15e4-bbd0-4719 a319-c731f79a94ab
Contained by
767a652b-8176-40b1 980a-71c5c48e37d6
succeeded by
preceded by
3d9a68c3-8594-4f1d 923b-44663c070bcc
similar
succeeded by
similar
preceded by
c44dde3a-2022-4517 bfaf-b2fc34b0d99a

Figure 1: Graph File System

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.

2.4 Interface

One of the lessons from Plan 9 is that everything can be represented as a file [73]; 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

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 3: Graph File Systems Operations

types [8, 85] 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 [84]; and 5) kernel extensions, e.g., provenance tracking systems [72] 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 2 might describe a file that came from a particular email or web site.

Figure 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.

2.5 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 [21] and attribute storage is largely a solved problem (although the last time one of the authors said that, her colleague disagreed [52]), so we focus on fast and efficient graph storage and query.

Today's systems either provide graph storage [61, 77, 97] or graph processing [26, 41, 45, 50, 66, 78, 82], 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 [48] meet all these requirements (we used them as the query and storage mechanism in the SHEEP graph partitioner [53]). 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.

3 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 [79] and AFS [83] 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 [87]. 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 [96]. The Google File System [24] 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* [11]: "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 [20] and faceted search [7, 14, 30, 40, 89, 94]. 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 [70], 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.

4 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 [4]. Similarly, Windows offer a similar extensible service [62]. 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 [1–3, 13, 23, 35, 42, 43, 46, 64, 71, 90]. Automatic tagging systems have become a more common approach here as manual tagging by users has proven to be impractical [84, 85]. The addition of *semantic* information [3, 15, 19, 22, 25, 27, 28, 31, 35, 49, 55–58, 65, 69, 71, 88, 95] 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] [80] .

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 [63, 72] 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 [24], 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 [3, 9, 29, 33, 39, 44, 51, 58, 64, 69, 92].

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

5 CONCLUSION

Hierarchical file systems have served us far longer than we had any right to expect over the past half century, but the need to find and realize a new paradigm becomes more critical each minute we spend searching for that elusive document. To date, our model for coping with this lack of usability has been to ignore the file system namespace entirely — is there any more strident statement as to the model's inadequacy than *irrelevance*?

There is a place for a modern file system name space in systems; we do not need to abrogate our responsibility to provide this traditional service by relying upon our programs and users to compensate for our dereliction of responsibility. The graph file system is one approach to address these concerns in a novel way; we claim that it better achieves the goals that have been set forth over the years by clearly identifying the missing element in prior proposals: capturing and exploiting *data relationships.* This is not so much a bold new model as it is a realization that our existing model is constrained, and we can better achieve our goals by using the time-honored systems tradition of relaxing constraints to achieve improved levels of functionality. By elevating arbitrary and generalized relationships as first class file system elements, we can provide a better user experience.

Some will avoid this work, because it touches upon the unpredictable and messy area of human users. However, it is time to realize that our systems are no longer our personal playground, but a critical service to humanity.

REFERENCES

[1] I. Amazon Web Services, "Object tagging," 2018, (accessed January 8, 2019). [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/dev/object-tagging.html

[2] 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.

[3] P. Andrews, I. Zaihrayeu, and J. Pane, "A classification of semantic annotation systems," *Semantic Web*, vol. 3, no. 3, pp. 223–248, 2012.

[4] Apple, "Search and spotlight," (accessed January 6, 2019). [Online]. Available: https://tinyurl.com/yasf7kf2

[5] R. Appuswamy, D. van Moolenbroek, and A. Tanenbaum, "Loris - a redundant array of independent physical layers," 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.

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

[7] 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*, vol. 37, pp. 55–74, 2016.

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

[9] R. Boardman, R. Spence, and M. A. Sasse, "Too many hierarchies? the daily struggle for control of the workspace," in *Proceedings of HCI international*, vol. 1, 2003, pp. 616–620.

[10] 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, pp. 49–60. [Online]. Available: https://www.usenix.org/conference/atc13/technical-sessions/presentation/bronson

[11] V. Bush *et al.*, "As we may think," *The atlantic monthly*, vol. 176, no. 1, pp. 101–108, 1945.

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

[13] J. Chou, "Findfs: adding tag-based views to a hierarchical filesystem," Ph.D. dissertation, University of British Columbia, 2015.

[14] P. H. Cleverley and S. Burnett, "Retrieving haystacks: a data driven information needs model for faceted search," *Journal of Information Science*, vol. 41, no. 1, pp. 97–113, 2015.

[15] V. Codocedo and A. Napoli, "Formal Concept Analysis and Information Retrieval – A Survey," *Formal Concept Analysis: 13th International Conference, ICFCA 2015*, pp. 61–77, 2015. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-19545-2{_}4

[16] 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, pp. 185–196.

[17] H. Custer, "Inside the windows nt file system," 1994.

[18] 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, pp. 213–229.

[19] 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, pp. 51–55.

[20] 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*, vol. 49, no. 2. ACM, 2016, pp. 28–35.

[21] 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, pp. 119–123.

[22] S. Faubel and C. Kuschel, "Towards semantic file system interfaces," *CEUR Workshop Proceedings*, vol. 401, 2008.

[23] O. Frieder and S. Kapoor, "Hierarchical structured abstract data organization system," June 2012, uS Patent 8,209,358.

[24] S. Ghemawat, H. Gobioff, and S.-t. Leung, "Google_File_System," 2003.

[25] D. K. Gifford, P. Jouvelot, M. A. Sheldon *et al.*, "Semantic file systems," in *ACM SIGOPS operating systems review*, vol. 25, no. 5. ACM, 1991, pp. 16–25.

[26] 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*, vol. 14, 2014, pp. 599–613.

[27] B. Gopal and U. Manber, "Integrating content-based access mechanisms with hierarchical file systems," in *OSDI*, vol. 99, 1999, pp. 265–278.

[28] 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, p. 1.

[29] 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, pp. 1125–1136.

[30] M. Hearst, "Design recommendations for hierarchical faceted search interfaces," in *ACM SIGIR workshop on faceted search*. Seattle, WA, 2006, pp. 1–5.

[31] 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*, vol. 27, no. 4, pp. 1212–1225, 2016.

[32] 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, pp. 3–12.

[33] 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*, vol. 7, no. 2, pp. 107–122, 2011.

[34] 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, pp. 767–776.

[35] 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." pp. 1–5, 2016.

[36] 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, pp. 1505–1508.

[37] D. R. Karger and W. Jones, "Data unification in personal information management," *Communications of the ACM*, vol. 49, no. 1, pp. 77–82, 2006.

[38] 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, pp. 2669–2678.

[39] 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, p. 543.

[40] 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, pp. 573–583.

[41] A. Kyrola, G. E. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a pc." USENIX, 2012.

[42] A. Laursen, "A novel, tag-based file-system," Master's thesis, Macalester College, 2014, (accessed March 29, 2018). [Online]. Available: https://digitalcommons.macalester.edu/mathcs_honors/34/

[43] 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.

[44] 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, p. 118.

[45] 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.

[46] 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, pp. 3745–3750.

[47] 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, pp. 124–128.

[48] 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.

[49] 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*, vol. 2003-Janua, pp. 115–121, 2003.

[50] 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, pp. 135–146.

[51] 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*, pp. 627–634, 1992. [Online]. Available: http://portal.acm.org/citation.cfm?doid=142750.143055

[52] 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, pp. 183–196.

[53] D. Margo and M. Seltzer, "A scalable distributed graph partitioner," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1478–1489, 2015.

[54] 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, pp. 122–129.

[55] 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*, pp. 56–63, 2005.

[56] B. Martin, "Formal concept analysis and semantic file systems," in *International Conference on Formal Concept Analysis*. Springer, 2004, pp. 88–95.

[57] ——, "Formal concept analysis and semantic file systems," Ph.D. dissertation, University of Wollongong, 2008. [Online]. Available: https://ro.uow.edu.au/theses/260/

[58] ——, "Open source libferris: Chasing the "everything is a file system" dream," *linux.com*, January 2014, (accessed September 4, 2018). [Online]. Available: https://www.linux.com/learn/open-source-libferris-chasing-everything-file-system-dream

[59] J. Masci, M. M. Bronstein, A. M. Bronstein, and J. Schmidhuber, "Multimodal similarity-preserving hashing," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 4, pp. 824–830, 2014.

[60] 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)*, vol. 2, no. 3, pp. 181–197, 1984.

[61] Microsoft, "Welcome to azure cosmos db," January 2019, (accessed January 17, 2019). [Online]. Available: https://docs.microsoft.com/en-us/azure/cosmos-db/introduction

[62] ——, "Data add-in interfaces," (accessed January 6, 2019). [Online]. Available: https://tinyurl.com/y83elc34

[63] 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*, ser. ATEC '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267359.1267363

[64] nayuki, "Designing better file organization around tags, not hierarchies," March 2017, (accessed October 1, 2018). [Online]. Available: https://www.nayuki.io/page/designing-better-file-organization-around-tags-not-hierarchies

[65] 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, pp. 139–142.

[66] 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, pp. 456–471.

[67] 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, pp. 89–104.

[68] M. A. Olson *et al.*, "The design and implementation of the inversion file system," in *USENIX Winter*, 1993, pp. 205–218.

[69] P. Omvlee, "A novel idea for a new Filesystem," *June*, 2009. [Online]. Available: http://referaat.cs.utwente.nl/conference/11/paper/6966/a-novel-idea-for-a-new-filesystem.pdf

[70] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: bringing order to the web." 1999.

[71] 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*, pp. 1–11, 2014.

[72] 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*, ser. SoCC '17. New York, NY, USA: ACM, 2017, pp. 405–418. [Online]. Available: http://doi.acm.org/10.1145/3127479.3129249

[73] 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, pp. 1–5.

[74] D. Reinsel, J. Gantz, and J. Rydning, "Data age 2025: The digitization of the world from edge to core," Seagate, Tech. Rep., November 2018, (accessed January 6, 2019). [Online]. Available: https://tinyurl.com/y95ogat4

[75] D. M. Ritchie and K. Thompson, "The unix time-sharing system," in *ACM SIGOPS Operating Systems Review*, vol. 7, no. 4. ACM, 1973, p. 27.

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

[77] M. Rudolf, M. Paradies, C. Bornhövd, and W. Lehner, "The graph story of the sap hana database." in *BTW*, vol. 13. Citeseer, 2013, pp. 403–420.

[78] 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, p. 22.

[79] 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.

[80] M. I. M. Seltzer and N. Murphy, "Hierarchical File Systems are Dead," *Applied Sciences*, p. 1, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855569{%}7B{%}25{%}7D5Cnhttps://www.usenix.org/event/hotos09/tech/full{%}7B{_}{%}7Dpapers/seltzer/seltzer.pdf

[81] 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, pp. 17–30. [Online]. Available: https://www.usenix.org/conference/fast13/technical-sessions/presentation/shetty

[82] J. Shun and G. E. Blelloch, "Ligra: a lightweight graph processing framework for shared memory," in *ACM Sigplan Notices*, vol. 48, no. 8. ACM, 2013, pp. 135–146.

[83] B. Sidebotham *et al.*, *Volumes: the andrew file system data structuring primitive*. Carnegie Mellon University, Information Technology Center, 1986.

[84] C. A. Soules and G. R. Ganger, "Why can't i find my files?: New methods for automating attribute assignment," in *HotOS*, 2003, pp. 115–120.

[85] ——, "Toward automatic context-based attribute assignment for semantic file systems," *Parallel data laboratory, Carnegie Mellon University. June*, 2004.

[86] 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*, ser. SOSP '73. New York, NY, USA: ACM, 1973, pp. 8–21. [Online]. Available: http://doi.acm.org/10.1145/800009.808043

[87] M. K. Sreenivas, S. R. Steiner, A. Brjazovski, and S. H. Agarwal, "Bypass of the namespace hierarchy to open files," Apr. 12 2011, uS Patent 7,925,681.

[88] M. Suguna and T. Anand, "Dynamic Metadata Management in Semantic File Systems," vol. 5, no. 3, pp. 44–47, 2015.

[89] D. Tunkelang, "Faceted search," *Synthesis lectures on information concepts, retrieval, and services*, vol. 1, no. 1, pp. 1–80, 2009.

[90] L. Up, "Tag , no di !" pp. 1–23, 2016.

[91] K. J. Vicente and R. C. Williges, "Accommodating individual differences in searching a hierarchical file system," *International Journal of Man-Machine Studies*, vol. 29, no. 6, pp. 647–668, 1988.

[92] 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, p. 587.

[93] 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*, vol. 12, 2015.

[94] ——, "Looking for answers: A usability study of online finding aid navigation," *The American Archivist*, vol. 80, no. 1, pp. 30–52, 2017.

[95] G. Wang, H. Lu, G. Yu, and B. YuBao, "Managing very large document collections using semantics," *Journal of Computer Science and Technology*, vol. 18, no. 3, pp. 403–406, 2003.

[96] G. Watumull, J. Gerend, L. Poggemeyer, A. Hansen, and K. Ainapure, "Resilient file system (refs) overview," (accessed January 17, 2019). [Online]. Available: https://tinyurl.com/y832s3ky

[97] J. Webber and I. Robinson, *A programmatic introduction to neo4j*. Addison-Wesley Professional, 2018.

[98] M. Wilkes, "A programmer's utility filing system," *The Computer Journal*, vol. 7, no. 3, pp. 180–184, 1964.