Percipience: Associative File Systems for Unstructured Data Relationships

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

Affiliation

I started my PhD in September 2017 at the University of British Columbia (UBC). I am co-supervised by three faculty at UBC: Norm Hutchinson (Computer Science), Sasha Fedorova (Electrical and Computer Engineering), and Andrew Warfield (Amazon and UBC Computer Science).

Abstract

File systems have evolved tremendously over the past six decades. However, that evolution has been lopsided: the *storage management* of file systems has evolved but the name space has not.

The hierarchical file system name space was proposed as part of Multics in 1965 and what they described is fundamentally what is used today. However, storage capacity has exploded — and continues to grow at a rate of 2.5 *zettabytes* per year. The vast majority of this is stored in general purpose file systems.

The failure of the namespace to evolve has made much of that data inaccessible to us. The Human Computer Interaction (HCI) community has been pointing out its inadequacies for *decades* yet the systems community has failed to expand the capabilities of file systems to enable the development of viable alternatives to an ever-growing pool of files.

Thus, the research question that I am exploring as the core of my PhD: how do we move beyond the current model and create a rich class of useful mechanisms to enable us to find what we are looking for in our ever-growing collection of files.

1 INTRODUCTION

> *Facts do not cease to exist because they are ignored.*Proper Studies
> Aldous Huxley (1927)

Just like facts, our files do not disappear simply because we choose not to find them. We have long since reached the point at which the volume, size, and diversity of our data is beyond the ability of most users to find the specific data they need.

Web-based search engines permit us to find a vast trove of useful information. However, it seems unlikely that users will want to publish their information publicly simply to allow web search indexing yet there is no economic model for indexing private user data at present.

Another approach is to construct search services, but they suffer from a myopic view of how we should search for files as they focus on *attributes* of our files, when also want to track and manage the *relationships* between our files. While we can certainly add some relationship information at the systems level — such as is done by provenance systems, for example — we do not have the full context of those relationships.

Similarly, if we, as systems researchers, pick a specific set of pre-defined relationships to support, it seems likely that we will choose incorrectly — we will not capture the vast array of useful relationships beyond our current ability to envision. Thus, we should explore ways in which we can permit users and their tools to create and manage those relationships; the file system then provides a vehicle for storing them, retrieving them, and permitting them to be explored. In that way we *enable* the broader development of better models for mining our data.

Prior work has established that a rich name space is of primary value to *users*. Applications do not benefit from hierarchical structured name spaces, as is demonstrated by non-hierarchical file systems optimized for distributed file systems [26] and key/value stores [6, 12, 13, 17, 18, 21, 25, 30, 33, 36, 41, 46, 51, 53, 56, 63, 65, 74, 94, 115, 117, 118, 120]. Indeed, having applications create content that users cannot directly access yet exist within the hierarchical name space slows down brute-force searching without yielding any additional utility.

Unlike applications, users need some form of structure for finding their data. Unfortunately, they are ill-served by the hierarchical name space as it forces them to limit them to tagging within the names of their files and directories.

Further confounding this is the division of data into "structured" and "unstructured": the former meaning data stored in a database, while the latter means data stored in a file system. The lines between these is not always clear: we have file systems implemented on top of databases [83], databases that provide storage of "semi-structured" data, culminating in the "data lake" [106], which looks surprisingly like a file system. In many cases, this "data lake" is *generated* data: log files, customer data, sales data. Often *business oriented* and while useful to some, it is not the ordinary users' interest.

If we instead consider a file system that manages *relationships* between our files, we realize that hierarchical file systems capture one relationship: the *contains* relationship.

Thus, my research question: *How can we transform operating systems to enable providing a generalized mechanism for tracking data relationships*? Driving this are related questions from other fields: what relationships are needed in order to provide enhanced search mechanisms for users? How do we enable the ability of a new class of tools that present user data in a way that is more effective for users? What services should we provide to applications to enable embedding relationship information?

I propose a possible *Graph File System* in §4. In §3 I describe work to construct a framework for exploring both compatibility and extensibility for file systems.

2 BACKGROUND

The concept of an hierarchical file system has existed for decades. An early Multics paper describes it in 1965 [16]. However, this early work also suggests the authors understood the hierarchical structure's limitations as they introduced their inclusion of the **link** concept, converting their tree into a graph.

Other operating systems adopted the hierarchical namespace model as well, including TENEX [78] and UNIX [89]. Ultimately, the hierarchical file system name space became the *de facto* norm.

The need to find information based upon content — not provided by the hierarchical name space file systems — is identified early on: in UNIX one of the 14 critical programs listed is the *permuted index server*. This is more remarkable given that most of the other tools were explicit *programming* tools [89].

Prior work argues the hierarchical structure is not workable for many users. The body of HCI research in this area is deep, showing that it works for, at most, a subset of people [93, 111, 112]

Thus, both the current structure of file systems information and the need to be able to find files based upon something *other than* the name of the file are long-standing concerns. As the number of files has grown, the challenges for simply finding existing material has also grown. Further aggravating the extant situation is the proliferation of the *types* of information that are also being produced. Each new application can potentially have its own new data format.

Prior file systems level work has focused primarily on a **semantic** model. Early work in this area proposed building *semantic file systems* [27], though there were certainly other more nuanced approaches considered [77]. This has become the approach to the problem in the web search world, where users are now familiar with and comfortable with a number of features (e.g., search) and behaviors (e.g., variable results from identical searches) [34, 73, 93, 96].

The systems community has echoed the HCI community in noting that hierarchical file systems have exceeded their useful life [95]. Despite this, the pragmatic reality is that any effective proposal to add functionality to the file systems requires compatibility. A number of past projects have focused on extending functionality on top of existing interfaces [14, 70, 85, 119].

A more interesting approach to the problem arises as you consider the problem akin to a *social graph*. While trees are a form of graph, by envisioning files as being vertices in a graph and the **relationships** between the files as edges we see a potentially new way of looking at file systems [19]. Graphs have an attractive expressive richness that is attractive: the existing hierarchical model is just a restricted graph. Interesting models, such as the source code tracking system **git**, use a graph to map the changes in files over time. More general models of provenance can be expressed as graphs as well [87, 99, 100, 108].

The need for better file system name spaces is well-established.

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 [5]. Similarly, Windows offer its own extensible service [75]. These enable searching based upon attributes including file suffix, date, size, etc., and context-sensitive content (such as music files by artist, composer, song title, or even *rights*). They do not provide the ability to search by relationship.

Tag Systems are an approach to improving hierarchical file systems searchability [2–4, 14, 24, 39, 48, 49, 57, 79, 85, 109]. Automatic tagging systems have become a more common approach here as manual tagging by users has proven to be impractical [102, 103]. The addition of *semantic* information [4, 15, 19, 23, 27, 29, 31, 34, 39, 59, 67, 69–71, 80, 84, 85, 105, 114] 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, prior work has pointed this out (perhaps subconsciously) by saying "How many of them [files] are *related* to each other?" [emphasis added] [95] .

3 FINESSE

In order to provide a framework upon which I could build a graph file system, I designed and implemented *Finesse*, an extension to FUSE that would enable supporting traditional POSIX file systems semantics via a kernel reflector, while at the same time easing the ability to extend the interface and thus enable the newer search functionality I seek to add. To do this I modified the user mode FUSE library to support

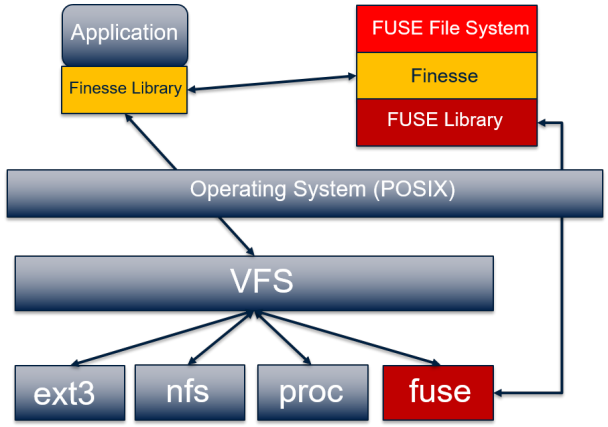

Figure 1: *Finesse* Architecture

operations from both the kernel mode driver as well as a new user-mode message passing interface. To ensure proper functionality, I focused on ensuring correct behavior even when the paths are mixed. Despite a large body of prior work, I could not find any that had attempted to use such a hybrid approach.

To simplify the use with application programs, I constructed a binary interposition mechanism using the Linux **LD_PRELOAD** mechanism, in which shared library calls are re-routed to a secondary implementation. This permitted me to work with existing application programs, without modification, as well as construct new applications that link against the Finesse implementations of standard POSIX I/O operations, and ultimately to utilize extended APIs.

This requires indirectly managing state, as supporting the mix of both message passing and system calls required keeping the relevant state in sync between the application, the kernel, and the FUSE library itself. This approach had the benefit of allowing existing FUSE file systems, as well as existing applications, to work without changes. I observed that a secondary benefit of this would be to explore improving the performance of FUSE, as prior work has observed that many meta-data heavy operations are expensive in the FUSE model [110].

I modified the existing FUSE file system implementation on Linux and augmented the low-level interface to handle requests from multiple sources. I used the existing unmodified **passthrough_ll** file system sample to evaluate my work.

Figure 1 provides a high level block diagram showing the constituent parts of *Finesse*. The key components are:

- ***Finesse* Application Library (finesse-app)** — this is the set of libraries that permit explicit invocation of Finesse functionality, or the implicit introduction of changes using **LD_PRELOAD**.
- ***Finesse* FUSE Extension (finesse-ext)** — this is the extensions I have added to FUSE.
- **FUSE Library (fuse-l)** — this is the existing FUSE library, used to construct FUSE file systems.
- **FUSE Kernel (fuse-k)** — this is the existing FUSE kernel driver, used to redirect file system calls to user mode.

In this model, existing applications work without any changes, including interception of calls. In this case, their behavior remains the same — as if *Finesse* were not on the system. Applications may employ passive augmented functionality via shared libraries, such as provided by **finesse-app**. Aware applications may also invoke new calls within the **finesse-app** libraries that offer enhanced functionality. If that functionality is not supported, the call is failed and the application may either fall back to compatibility mode or terminate operation, as appropriate.

Similarly, FUSE file systems may continue to support the existing interfaces *without modification*. In that case, only extensions added to *Finesse* — through **finesse-ext** — that rely upon existing FUSE interface functionality would be available. For experimental file systems, additional calls can be added to the existing FUSE function dispatch table, and invoked by **finesse-ext** as appropriate.

4 GRAPH FILE SYSTEM

My proposed file system focuses on *relationships* between my files. I 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, I treat directories as the embodiment of the *contains* relationship, not objects.

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

4.1 File System as a Graph Database

In considering a graph based file system, first consider implementing a file system in a graph database, of which there are many (§2). Their primary focus is the storage of graph-structured *data*. My 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

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

Relationship	Description
similar	Similarity measure, e.g., [72]
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

file systems. Still, others with compelling reason may wish to explore that option.

4.2 File System as Social Network

Next, 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 [9]. 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 [83] and other index-based structures [97] to store files, the community seems to have concluded that such storage is not ideal. I agree and suggest 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 I use a single graph-structured representation both in persistent store and in-memory. I 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 my relationship-centric file system, I present a new model and file system design.

4.3 Graph FS Model

I set out a basic description of my core objects in Table 1 and a demonstrative set of example relationships in Table 2. I do not consider either of these to be exhaustive, but rather propose them as an initial basis for discussion. The model I present 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 number or object ID. I do not rely upon *names* as they are simply mutable properties.

A *relationship* is a directional association between two files. I expect there to be far fewer relationships than files, though many more *instances* of relationships (i.e., the number of edges in my graph exceeds the number of vertices). Relationships may be either uni-directional (e.g., derived from) or bi-directional (e.g., similar). Table 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.

4.4 Interface

One of the lessons from Plan 9 is that everything can be represented as a file [88]; I expect to continue with this paradigm as it has served us well over the years. While I 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 my model of separating namespace from storage; how the storage providers return data to us is orthogonal to the namespace I use to retrieve it. For example, the *procfs* file system creates a synthetic namespace and supports I/O

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

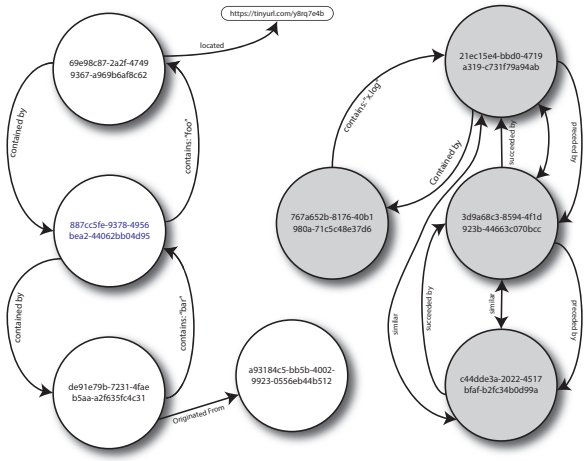

Figure 2: Graph File System

operations for reading and modifying data contents of the pseudo-files.

From the namespace perspective, my 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, I need the ability to remove each of these.

This 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, this mechanism can be used to create clusters of related files by asking for all the vertices connected by a specific relationship.

Where do relationships, labels, and properties come from? I 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 [7, 103] will produce more attributes; 3) applications will generate both attributes and relationships; 4) users may generate attributes and relationships, although history suggests that requiring users annotate their data is not effective [102]; and 5) kernel extensions, e.g., provenance tracking systems [86] 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 2 provides a simplified visualization of my graph file system model. My inclusion of disjoint graphs captures the notion that the system naturally supports multiple namespaces, implemented as disconnected graph components.

4.5 Aspects of Implementation

In considering a full implementation of the proposed graph file system, I suggest novel potential ways in which I can accomplish it. Using an object store [22] for files and attributes isolation is largely a solved problem, though some disagree [63]. This allows me to focus on fast and efficient graph storage and query.

Today's systems either provide graph storage [76, 90, 116] or graph processing [28, 47, 55, 60, 81, 91, 98], but a graph file system needs a high performance, space-efficient, mutable and queryable native graph representation. Prior work has found that mutable compressed sparse row representations [58, 64] meet all these requirements. 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.

5 CONCLUSION

The classic model of a file system as a simple dumb store of "unstructured" information fails to provide a manageable model for using and finding the vast trove of personal data that it accumulates; this is true whether we consider it from a strictly local perspective, or if we view it more generally and consider the broad range of locations where personal files are now stored, such as the cloud and mobile devices.

Thus, the existing paradigm has broken down. Users are given almost unlimited storage, which in turn encourages saving everything. *Finding* things is often done by search and identify or brute force search mechanisms. For cases of content that was externally obtained, it is often easier to simply re-download it than to find where it is presently stored.

My research focus is exploring how we can improve operating systems to provide richer semantics for identifying, finding, and searching for file content based upon file associations as an attribute clustering mechanism. While the exact scope of the final research remains to be determined, the emphasis on enabling applications and ultimately users to provide new ways of finding personal data is the core of this research direction.

REFERENCES

[1] 7dayshop.com. The evolution of a terabyte of data: 1956 –– 2014, 2015. (accessed October 1, 2018).

[2] Amazon Web Services, I. Object tagging, 2018. (accessed January 8, 2019).

[3] Ames, S., Bobb, N., Greenan, K. M., Hofmann, O. S., Storer, M. W., Maltzahn, C., Miller, E. L., and Brandt, S. A. 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* (2006), IEEE.

[4] Andrews, P., Zaihrayeu, I., and Pane, J. A classification of semantic annotation systems. *Semantic Web 3*, 3 (2012), 223–248.

[5] Apple. Search and spotlight. (accessed January 6, 2019).

[6] Berezecki, M., Frachtenberg, E., Paleczny, M., and Steele, K. Many-core key-value store. In *Green Computing Conference and Workshops (IGCC), 2011 International* (2011), IEEE, pp. 1–8.

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

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

[9] Bronson, N., Amsden, Z., Cabrera, G., Chakka, P., Dimov, P., Ding, H., Ferris, J., Giardullo, A., Kulkarni, S., Li, H., Marchukov, M., Petrov, D., Puzar, L., Song, Y. J., and Venkataramani, V. 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, 2013), USENIX, pp. 49–60.

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

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

[12] Chandramouli, B., Prasad, G., Kossmann, D., Levandoski, J., Hunter, J., and Barnett, M. Faster: A concurrent key-value store with in-place updates. In *Proceedings of the 2018 ACM International Conference on Management of Data* (2018), ACM.

[13] Chen, X., Sha, E. H.-M., Abdullah, A., Zhuge, Q., Wu, L., Yang, C., and Jiang, W. Udorn: A design framework of persistent in-memory key-value database for nvm. In *Non-Volatile Memory Systems and Applications Symposium (NVMSA), 2017 IEEE 6th* (2017), IEEE, pp. 1–6.

[14] Chou, J. *FindFS: adding tag-based views to a hierarchical filesystem.* PhD thesis, University of British Columbia, 2015.

[15] Codocedo, V., and Napoli, A. Formal Concept Analysis and Information Retrieval – A Survey. *Formal Concept Analysis: 13th International Conference, ICFCA 2015* (2015), 61–77.

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

[17] Das, S., Agrawal, D., and El Abbadi, A. G-store: a scalable data store for transactional multi key access in the cloud. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 163–174.

[18] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS operating systems review* (2007), vol. 41, ACM, pp. 205–220.

[19] Di Sarli, D., and Geraci, F. Gfs: a graph-based file system enhanced with semantic features. In *Proceedings of the 2017 International Conference on Information System and Data Mining* (2017), ACM, pp. 51–55.

[20] Eck, O., and Schaefer, D. A semantic file system for integrated product data management. *Advanced Engineering Informatics 25*, 2 (2011), 177–184.

[21] Escriva, R., Wong, B., and Sirer, E. G. Hyperdex: A distributed, searchable key-value store. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication* (2012), ACM, pp. 25–36.

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

[23] Faubel, S., and Kuschel, C. Towards semantic file system interfaces. *CEUR Workshop Proceedings 401* (2008).

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

[25] Geambasu, R., Levy, A. A., Kohno, T., Krishnamurthy, A., and Levy, H. M. Comet: An active distributed key-value store. In *OSDI* (2010), pp. 323–336.

[26] Ghemawat, S., Gobioff, H., and Leung, S.-t. Google_File_System.

[27] Gifford, D. K., Jouvelot, P., Sheldon, M. A., et al. Semantic file systems. In *ACM SIGOPS operating systems review* (1991), vol. 25, ACM, pp. 16–25.

[28] Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., and Stoica, I. Graphx: Graph processing in a distributed dataflow framework. In *OSDI* (2014), vol. 14, pp. 599–613.

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

[30] Guo, P. J., and Seltzer, M. Burrito: Wrapping your lab notebook in computational infrastructure. In *Proceedings of the 4th USENIX Workshop on the Theory and Practice of Provenance* (Berkeley, CA, USA, 2012), TaPP'12, USENIX Association.

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

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

[33] Hu, X., Wang, X., Li, Y., Zhou, L., Luo, Y., Ding, C., Jiang, S., and Wang, Z. Lama: Optimized locality-aware memory allocation for key-value cache. In *USENIX Annual Technical Conference* (2015), pp. 57–69.

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

[35] Hua, Y., Jiang, H., Zhu, Y., and Feng, D. Rapport: Semantic-sensitive Namespace Management in Large-scale File Systems. *CSE Technical reports* (2010).

[36] Huang, K., Zhou, J., Huang, L., and Shen, Y. Nvht: An efficient key-value storage library for non-volatile memory. *Journal of Parallel and Distributed Computing* (2018).

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

[38] Jensen, C., Lonsdale, H., Wynn, E., Cao, J., Slater, M., and Dietterich, T. G. The life and times of files and information: a study

of desktop provenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2010), ACM, pp. 767–776.

[39] Jones, I., You, W., and Do, C. A. N. Tagxfs is a semantic file system. It extends the user space file system to a tag based hierarchy. 1–5.

[40] Jones, W., Phuwanartnurak, A. J., Gill, R., and Bruce, H. Don't take my folders away!: organizing personal information to get ghings done. In *CHI'05 extended abstracts on Human factors in computing systems* (2005), ACM, pp. 1505–1508.

[41] Kalia, A., Kaminsky, M., and Andersen, D. G. Using rdma efficiently for key-value services. *ACM SIGCOMM Computer Communication Review 44*, 4 (2015), 295–306.

[42] Karger, D. R., and Jones, W. Data unification in personal information management. *Communications of the ACM 49*, 1 (2006), 77–82.

[43] Kärle, E., Şimşek, U., and Fensel, D. semantify. it, a platform for creation, publication and distribution of semantic annotations. *arXiv preprint arXiv:1706.10067* (2017).

[44] Karlson, A. K., Smith, G., and Lee, B. 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* (2011), ACM, pp. 2669–2678.

[45] Khan, M. T., Hyun, M., Kanich, C., and Ur, B. 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* (2018), ACM, p. 543.

[46] Kim, J., Lee, S., and Vetter, J. S. Papyruskv: a high-performance parallel key-value store for distributed nvm architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2017), ACM, p. 57.

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

[48] Laursen, A. A novel, tag-based file-system. Master's thesis, Macalester College, 2014. (accessed March 29, 2018).

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

[50] Leung, A., Parker-Wood, A., and Miller, E. Copernicus: A scalable, high-performance semantic file system. *University of California, Santa …*, October (2009).

[51] Lim, H., Han, D., Andersen, D. G., and Kaminsky, M. Mica: A holistic approach to fast in-memory key-value storage. USENIX.

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

[53] Liu, H., Huang, L., Zhu, Y., and Shen, Y. Librekv: A persistent in-memory key-value store. *IEEE Transactions on Emerging Topics in Computing* (2017).

[54] Liu, X., Niv, G., Shenoy, P., Ramakrishnan, K., and Van der Merwe, J. The case for semantic aware remote replication. *Proceedings of the second ACM workshop on Storage security and survivability - StorageSS '06* (2006), 79.

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

[56] Lu, G., Nam, Y. J., and Du, D. H. Bloomstore: Bloom-filter based memory-efficient key-value store for indexing of data deduplication on flash. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on* (2012), IEEE, pp. 1–11.

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

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

[59] Mahalingam, M., Tang, C., and Xu, Z. Towards a semantic, deep archival file system. *Proceedings of the IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems 2003-Janua* (2003), 115–121.

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

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

[62] Mander, R., Salomon, G., and Wong, Y. Y. A "Pile" Metaphor for Supporting Casual Organization of Information. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92* (1992), 627–634.

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

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

[65] Marmol, L., Sundararaman, S., Talagala, N., and Rangaswami, R. Nvmkv: A scalable, lightweight, ftl-aware key-value store. In *USENIX Annual Technical Conference* (2015), pp. 207–219.

[66] Marsden, G., and Cairns, D. E. 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* (2003), South African Institute for Computer Scientists and Information Technologists, pp. 122–129.

[67] Martin, B. Formal concept analysis and semantic file systems. In *International Conference on Formal Concept Analysis* (2004), Springer, pp. 88–95.

[68] Martin, B. Formal concept analysis and semantic file systems. *Concept Lattices* (2004).

[69] Martin, B. *Formal concept analysis and semantic file systems*. PhD thesis, University of Wollongong, 2008.

[70] Martin, B. Open source libferris: Chasing the "everything is a file system" dream. *linux.com* (January 2014). (accessed September 4, 2018).

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

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

[73] Méndez, E., and Greenberg, J. Linked data for open vocabularies and hive's global framework. *El profesional de la información 21*, 3 (2012), 236–244.

[74] Menon, P., Rabl, T., Sadoghi, M., and Jacobsen, H.-A. Cassandra: An ssd boosted key-value store. In *2014 IEEE 30th International Conference on Data Engineering (ICDE)* (2014), IEEE, pp. 1162–1167.

[75] Microsoft. Data add-in interfaces. (accessed January 6, 2019).

[76] Microsoft. Welcome to azure cosmos db, January 2019. (accessed January 17, 2019).

[77] Mogul, J. C. *Representing information about files*. PhD thesis, Stanford University, 1986.

[78] Murphy, D. L. Storage organization and management in tenex. In

Proceedings of the December 5-7, 1972, fall joint computer conference, part I (1972), ACM, pp. 23–32.

[79] nayuki. Designing better file organization around tags, not hierarchies, March 2017. (accessed October 1, 2018).

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

[81] Nguyen, D., Lenharth, A., and Pingali, K. A lightweight infrastructure for graph analytics. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), ACM, pp. 456–471.

[82] Odom, W., Sellen, A., Harper, R., and Thereska, E. Lost in translation: understanding the possession of digital things in the cloud. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2012), ACM, pp. 781–790.

[83] Olson, M. A., et al. The design and implementation of the inversion file system. In *USENIX Winter* (1993), pp. 205–218.

[84] Omvlee, P. A novel idea for a new Filesystem. *June* (2009).

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

[86] Pasquier, T., Han, X., Goldstein, M., Moyer, T., Eyers, D., Seltzer, M., and Bacon, J. Practical whole-system provenance capture. In *Proceedings of the 2017 Symposium on Cloud Computing* (New York, NY, USA, 2017), SoCC '17, ACM, pp. 405–418.

[87] Pasquier, T., Singh, J., Powles, J., Eyers, D., Seltzer, M., and Bacon, J. Data provenance to audit compliance with privacy policy in the internet of things. *Personal and Ubiquitous Computing 22*, 2 (2018), 333–344.

[88] Pike, R., Presotto, D., Thompson, K., Trickey, H., and Winterbottom, P. 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* (1992), ACM, pp. 1–5.

[89] Ritchie, D. M., and Thompson, K. The unix time-sharing system. In *ACM SIGOPS Operating Systems Review* (1973), vol. 7, ACM, p. 27.

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

[91] Salihoglu, S., and Widom, J. Gps: a graph processing system. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management* (2013), ACM, p. 22.

[92] Schandl, B. Representing linked data as virtual file systems. *CEUR Workshop Proceedings 538* (2009).

[93] Schandl, B., and Haslhofer, B. 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) 5554 LNCS* (2009), 51–65.

[94] Schütt, T., Schintke, F., and Reinefeld, A. Scalaris: reliable transactional p2p key/value store. In *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG* (2008), ACM, pp. 41–48.

[95] Seltzer, M. I. M., and Murphy, N. Hierarchical File Systems are Dead. *Applied Sciences* (2009), 1.

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

[97] Shetty, P. J., Spillane, R. P., Malpani, R. R., Andrews, B., Seyster, J., and Zadok, E. 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, 2013), USENIX, pp. 17–30.

[98] Shun, J., and Blelloch, G. E. Ligra: a lightweight graph processing framework for shared memory. In *ACM Sigplan Notices* (2013), vol. 48, ACM, pp. 135–146.

[99] Simmhan, Y. L., Plale, B., and Gannon, D. A survey of data provenance in e-science. *ACM Sigmod Record 34*, 3 (2005), 31–36.

[100] Smith, W., Moyer, T., and Munson, C. Curator: Provenance management for modern distributed systems. *arXiv preprint arXiv:1806.02227* (2018).

[101] Soules, C., and Ganger, G. Toward automatic context-based attribute assignment for semantic file systems. *Parallel data laboratory, Carnegie Mellon …*, June (2004).

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

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

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

[105] Suguna, M., and Anand, T. Dynamic Metadata Management in Semantic File Systems. 44–47.

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

[107] Thereska, E., Riva, O., Banks, R., Lindley, S., Harper, R., and Odom, W. Beyond file systems: understanding the nature of places where people store their data. Tech. rep., Microsoft Research, 2013. (accessed March 29, 2018).

[108] Ujcich, B. E., Bates, A., and Sanders, W. H. A provenance model for the european union general data protection regulation. In *International Provenance and Annotation Workshop* (2018), Springer, pp. 45–57.

[109] Up, L. Tag , no di ! 1–23.

[110] Vangoor, B. K. R., Tarasov, V., and Zadok, E. To fuse or not to fuse: Performance of user-space file systems. In *FAST* (2017), pp. 59–72.

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

[112] Vicente, K. J., and Williges, R. C. Accommodating individual differences in searching a hierarchical file system. *International Journal of Man-Machine Studies 29*, 6 (1988), 647–668.

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

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

[115] Wang, P., Sun, G., Jiang, S., Ouyang, J., Lin, S., Zhang, C., and Cong, J. An efficient design and implementation of lsm-tree based key-value store on open-channel ssd. In *Proceedings of the Ninth European Conference on Computer Systems* (2014), ACM, p. 16.

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

[117] Wu, C., Faleiro, J. M., Lin, Y., and Hellerstein, J. M. Anna: A kvs for any scale.

[118] Wu, X., Xu, Y., Shao, Z., and Jiang, S. Lsm-trie: an lsm-tree-based ultra-large key-value store for small data. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference* (2015), USENIX Association, pp. 71–82.

[119] Xu, L., Huang, Z., Jiang, H., Tian, L., and Swanson, D. 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 (2014), 25–30.

[120] Zhang, W., Xu, Y., Li, Y., Zhang, Y., and Li, D. Flame db: A key-value store with grouped level structure and heterogeneous bloom filter. *IEEE ACCESS 6* (2018), 24962–24972.