File-Less Approach to Large Scale Data Management

Bartosz Kryza(✉) and Jacek Kitowski

Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications,
AGH University of Science and Technology, Krakow, Poland
bkryza@agh.edu.pl

Abstract. With the continuously increasing amount of online resources and data such use cases as discovery, maintenance and inter-operation become more and more complex. In particular, data management is becoming one of the main issues with respect to both scientific (large scale simulations or data mining applications) as well as consumer use cases (accessing photos or email attachments on mobile devices). We believe that one of the main bottlenecks blocking development of solutions providing truly seamless developer and user experience is the concept of file and filesystem. We present Filess, vision and architecture of file-less information systems where files are not necessary, neither in the application nor operating system layers.

Keywords: File systems · Data management · Hypergraphs

1 Introduction

Files and filesystems have been part of computer systems since the times of punch cards, stored in filing cabinets. Due to the technology used in the early days of computing such as magnetic tapes and until recently magnetic hard drives, files provided efficient way to store data in sequences of blocks which could be read from such media into memory. However, even then several researchers raised various problems related to such data storage [8,18,19]. In our opinion the main problems of modern IT systems related to files and filesystems are the following:

- *Data is unnaturally clustered into files* - once data item is stored in a file it becomes locked in this file, whether or not it is actually a part of a larger data structure or could be accessible on its own (consider for instance an image in a presentation, or a tag in an XML document),
- *Very large unnecessary data redundancy* - file based data management results in very large duplication of data due to the necessity of including data directly inside the file contents instead of referencing it (again images in presentations and rich text documents, attachments in emails, etc.) [7,10]. Existing files and filesystems do not provide means for uniform referencing of other files on a global scale, a feature which is the basis of WWW in the form of hyper links,

S. Hunold et al. (Eds.): Euro-Par 2015 Workshops, LNCS 9523, pp. 27–38, 2015.
DOI: 10.1007/978-3-319-27308-2_3

- *Inflexible hierarchical namespaces* - although it was convenient to store files in tree based directory structures when users had hundreds of files, with tens or hundreds of thousands of files it is impossible to memorize where a given file could be found without global filesystem indexing tools such as Spotlight, Tracker or Search Charm, which however only match queries to files names or using string based search over textual documents contents. No semantic search can be achieved based on relations between elements contained in these files (for instance *Find all images included in this paper*),
- *Barrier for the operating system* - it is impossible to address a specific piece of data inside the file from the level of the operating system (for instance for the use from the command line), thus it is in general impossible to get metadata about an image or music file, an application or specific library has to be executed to extract it, which are specific for each file format,
- *Lack of versioning support* - versioning of data can be achieved only by storing new files under different names (or in some cases storing text or binary differences between version, in application specific ways).

Let's consider for instance a single file such as a text document, presentation or even simple e-mail, which usually contains large amount of information which is lost in the structure of the file and thus not visible for querying by computers or even humans. For example, a corporate document prepared within some organization by several people contains several independent items (tables, charts, diagrams, paragraphs of text) of which each can have distinct author, different authorization policies and can be used in several other documents, while currently all this provenance information is lost once the document is saved as a file. The first problem is that for instance some figure from this document will be duplicated in every copy of this document and also in every new document that will use this figure. Additionally the information provided within the document itself about the picture brings very little information to the reader and no information at all that could be processed by computers. The main hypothesis of this research is that in order to make information actually reusable into knowledge in a world wide distributed setting, information must be stored in a way that it can be freely shared, reused and processed.

We propose to address this issue by introducing an architecture for information systems which departs from the concept of file and filesystem completely. We introduce a flexible and scalable data model based on hypergraphs, where data objects are stored in nodes and all relations are represented through hyperedges (edges which can connect more than 2 nodes). The hypergraph is provided to applications and operating systems via the same abstraction layer, which hides the actual storage system used as well as the fact of data distribution between devices. This paper refines the initial vision and requirements defined in our previous work [12,13].

2 Related Work

Most research in the area of making the existing directory based file systems more flexible can be classified into the area of semantic file systems [8], i.e.

file systems where files have attached meaning. This paper sketches a vision of file systems where files can be annotated in some way, and the basic file system operation such as copy or delete don't take directory paths as arguments but the *semantic* description of the files. The problem with these solutions is that still all the information is either fragmented or clustered into files, and the semantics deal only with meta data attached to these files in the form of some attributes. Nevertheless, these solutions are very important for our work as these approaches address important issues, mainly of how information can be found in file based systems. One of the formal attempts at file system implementation based on set theoretical basis is a file system using Formal Concept Analysis [5], which employs the FCA formal model of classification, neighborhood estimation and Boolean querying. A similar approach, although still bounded by the constraints of regular files, is the Logical File System project [16]. The basic role of this file system is to allow searching for files using first-order logic formulas instead of conventional directory paths. Unfortunately the use of first-order logic inference can seriously impair the scalability of the system in highly distributed settings. Until now, one major industrial attempt at abstracting the file concept from the operating system was the WinFS (Windows File System), which is a research effort from Microsoft [9]. Its basic assumption is to store all information about data in the system, including what would usually be referred to as file in a relational database. Furthermore, on the low level of storage device controllers, there is a trend to move from block device based interfaces (i.e. supporting file oriented systems) towards more flexible solutions such as OSB (Object-based Storage Device) [1], where instead of storing data in fixed size chunks the data can be stored in custom clusters of data along with relevant meta data. Unfortunately, most operating system level approaches still use these devices to store files, even if more efficiently [22,23]. However, with removal of the concept of the file all together, this approach will be a significant factor along with further adoption of SSD storage [17]. In fact Seagate has introduced recently an actual network attached object based device called Kinetic Storage [21], which provides a hardware back end for object based databases without any file system protocol access. Furthermore, certain technology enablers are emerging which provide insight into how the future storage could be improved on the low level. These include for instance various NVRAM (Non-Volatile RAM) solutions in particular memristor [24]. For prototype development an interesting solution is SanDisk's UlltraDIMM SSD [20], which is an SSD storage in the form of DIMM memory units, which can physically replace computers RAM memory. As we can see, there exists already several approaches and basic technologies which can support the proposed research concept. However, none of the existing solutions addresses abandoning the concept of a file as a whole, including all its repercussions on the storage, operating system, application and user interface level.

3 Filess Vision

Since its emergence, Cloud computing has become the leading paradigm in computing. The main reason for this was the fact that users found always-online

resources to be much easier and efficient to use. Here resources include computing services, web pages and data. This is also one of the reasons why Cloud storage services such as Dropbox or Google Drive [4] are so popular, i.e. users are mobile and have multiple devices and need access to their content wherever they are. However, all these services have to be built on top of existing operating systems, since none of the mainstream OS has support for such functionality. In fact most operating systems do not have any artifacts for supporting such scenarios as over the network data access, process migration or check-pointing, which would enable developers to provide users with truly seamless experience when using multiple devices, such as working on a single file using multiple applications on different devices simultaneously. Imagine for instance creation of a simple conference presentation. It consists of some slides with text, images, equations sometimes embedded movies. Whenever an image needs to be updated, it has to be done in a separate application, saved into a file and imported into the selected slide in the presentation editor. If the user wants to preview the presentation on her tablet, it needs to be transferred manually there using yet another application. In our vision all these applications would operate on a global data space, managed entirely by the Filess middleware. Thus an image changed in a photo editing application, would make the new image version automatically updated in the presentation and whenever the contents of the presentation had been modified, they would be instantly visible on the presentation preview on users tablet. Then, when the presentation is ready, all the user needs to know in order to present it during the talk is to know the ID of the root node in the graph data model representing the presentation.

Overview of basic assumptions and requirements for this work, as discussed in our previous work [13], is presented below:

- There are no files - neither in the storage, middleware, operating system or user interface layers. Of course, at the prototype stage such approach would be very expensive in order to remove files completely from existing operating systems which use files even for communication with hardware devices,
- Documents, E-mails, images, movies, web pages and all other concepts, which are in practice today synonyms for files, in our architecture are only manifestations/renderings of interconnected groups of objects shown to the user in a context dependent way,
- Data and meta data exist at the same level - for instance there is no difference between the *Image* object and the object describing its author or authorization policy - we do not plan to introduce a meta data mechanism such as Dublin Core or even Semantic Web,
- Data and information replication should be controlled by the middleware - it is not necessary for users to copy and store the information for either security or efficiency reasons. As a consequence data redundancy can be optimized by the middleware,
- The proposed approach inherently supports the ubiquitous computing paradigm i.e., there is no *Load document*, *Save document* operations. It is possible to work on a laptop, then literally just shut it down and switch to pocket

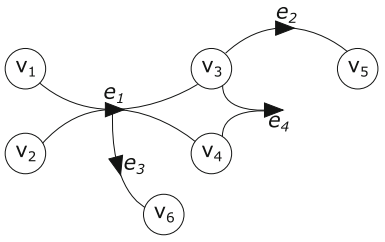
PC or mobile phone and all the changes will be seamlessly available there, assuming of course network access is omnipresent,
– Security, especially authorization is intertwined within the global information space along with the information itself, i.e. security assertions (and any *annotations* for that matter) are first class objects in the infrastructure.

4 Filess Data Model

The basic assumption of the underlying data model is that all data are stored as objects in the nodes of a hypergraph structure. Hypergraphs provide flexible and scalable data model, where data objects are stored in nodes and all relations are represented through hyperedges (edges which can connect more than 2 nodes). The hypergraph is provided to applications and operating systems via the same abstraction layer, which hides the actual storage system used as well as the fact of data distribution between devices. Hypergraphs enable more natural modeling of n-m relationships and modeling of objects with multiple properties using smaller number of edges. It has been shown, that hypergraphs enable modeling various common data models such as relational model, XML, JSON or even Semantic Web standards such as OWL [11].

4.1 Hypergraphs

Hypergraphs have been studied and applied in various areas of computer science [2,3,6]. Basically a hypergraph is a tuple $H = (V, E)$ where $V = \{v_1, v_2, v_3, \ldots, v_n\}$ is the set of vertices and $E = \{e_1, e_2, e_3, \ldots, e_m\}$ is the set of hyperedges. The main difference and generalization over regular graphs is that a hypergraph edges, called hyperedges, can connect any number of vertices. In case of undirected hypergraphs hyperedges are simple subsets of the power set of V i.e. $E \subset 2^V$. In case of directed hypergraphs, edges are tuples composed



(a) Hypergraph

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ e_1 \\ e_2 \\ e_3 \\ e_4 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{pmatrix} -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

(b) Incidence matrix

Fig. 1. Example of directed hypergraph

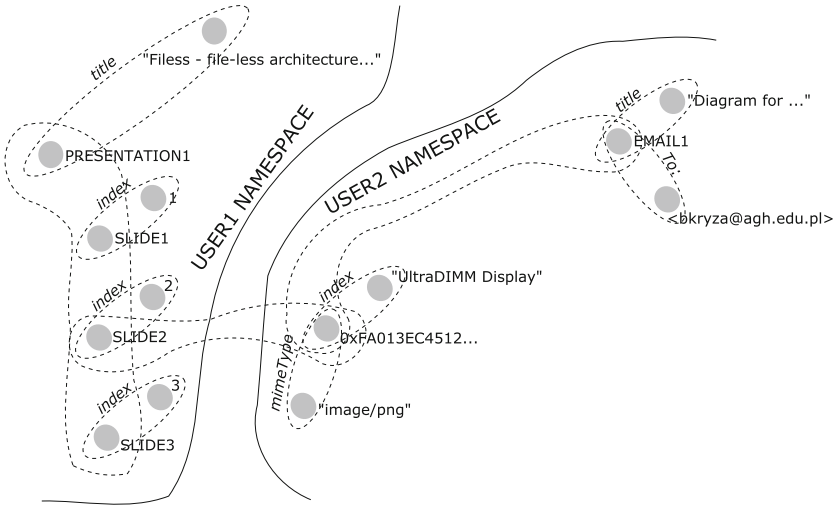


Fig. 2. Filess data model example

of *head* and *tail* sets: $E = (T_E, H_E)$. Hyperedges in general can connect both vertices and other edges. In our model, we assume that edges can appear only in the tail set of the edge (i.e. they can only be used to assign attributes through other edges to vertices. Thus, $T_E \subset V \cup E$ and $H_E \subset V$. We will also define the index sets for vertices and edges of graph H as $I_V(H)$ and $I_E(H)$ respectively. Let's consider an example graph in Fig. 1a and it's incidence matrix Fig. 1b. In the incidence matrix, edges are also as rows in order to model edges which have properties themselves (e.g. e3 on the graph). The example graph can be defined as:

$$H = (\{v_1, v_2, v_3, v_4, v_5, v_6\}, \{e_1 = (\{v_1, v_2\}, \{v_3, v_4\}), \\ e_2 = (\{v_3\}, \{v_5\}), e_3 = (\{e_1\}, \{v_6\}), e_4 = (\{v_3, v_4\}, \emptyset)\})$$

The usage of hypergraphs as the underlying formalism for the proposed data model allows us to reuse a large number of theorems and algorithms for their processing and validation.

4.2 Overview

First of all, all data in Filess is stored in data objects which are values assigned to the hypergraph vertices. Edges provide means for creating named relations and attributes between vertices. Leveraging the hypergraph property of allowing multi-vertice edges, it is natural to create n-m relations between data objects. An important aspect of the proposed data model is that of *namespaces*, which provide means for separating the global data object graph into subgraphs, which can intersect, i.e. each data object can belong to multiple namespaces simultaneously. Namespaces are an important part of the model as they provide means

for optimizing search and discovery of data in the distributed graph as well as enable basic security. The most important namespace is the user's namespace which is created automatically for each new user after login, which uniquely identifies users objects and relations. Furthermore, namespaces can be assigned to edges in order to provide means for disambiguating their meaning, in a form typical for existing on the web URI. An important notion is that namespaces are flat, i.e. there is no namespace hierarchy as in the case of filesystem directory structure. Namespaces can be considered as system level attributes which can be assigned to any element of the global hypergraph. The uniqueness of namespaces is achieved by using GUID's to define them, while each namespace can have multiple user-friendly names (for instance in various languages). Once a new actor (user or service) joins the global Filess model, the initial node with the actor identity is created in a new namespace. From then on, the data objects can be connected with data objects of other actors through global relations. An overview of this approach is presented in Fig. 2. Each data object can store basic types of values or be a composite object which only contains connections to other data objects:

- *Number* - this is a union data type which allows to store any numeric data type while providing users with a simple API, which handles actual data type identification on the library level,
- *String* - this data object provides means for storing any text in UTF8,
- *List* - most graph data modeling frameworks do not provide lists or arrays, which can be very inefficient when modeling using graph nodes. This data object provides a simple means for compositing a set of data objects into an ordered structure,
- *Binary* - this data object provides means for storing large binary data such as videos, where the actual data is hashed and stored in a separate distributed key-value store,
- *Composite* - composite data objects are objects which do not need to store any actual value in their node, but provide links for other data objects. Any object containing a value can also be a Composite object, in which case the value represents a flattened representation of the objects structure. This situation can occur during decomposition of an object into a graph,
- *Stream* - buffer objects provide abstraction over I/O functionality of the operating system, these objects cannot be transferred between nodes, and are volatile, i.e. their state and value cannot be synchronized and no version information for these objects is maintained, only read or write operations are allowed. These objects enable complete removal of file and filesystem concepts from the applications code.

4.3 Object Composition and Decomposition

The most important operations on data objects from the point of view of the abstract model are composition and decomposition.

Composition. Data object composition provides natural way for creating structure in the data model using hyperedges connecting various data objects, thus

enabling data discovery and graph traversal on the middleware level. Any set of data objects can be composed into other composite data objects using these operations without affecting or unnecessarily replicating the referenced data objects. Formally, composition transforms the initial hypergraph $H = (\{v_i\}_{i \in I_V(H)}, \{e_j\}_{j \in I_E(H)})$ as follows:

$$H' = (\{v_i\} \cup \{v'\}, \{e_j\} \cup \{(\{v'\}, \{v_k\}_{k \in I_V(H)})\})$$

i.e. it always extends the vertex set with one element (the new composite data object, v') and adds any number of necessary edges connecting vertices from the initial graph H.

Decomposition. Respectively, each data object can be decomposed into data objects which introduce structure into otherwise flat data object values. For instance consider data object containing a string value `"John Smith"`. This object could be decomposed using 2 edges: (`_:firstName`, `"John"`) and (`_:lastName`, `"Smith"`). However, since the object already existed in the previous form, some applications might rely on its flat representation thus it should remain in the Composite object after decomposition. In the future we are planning to enable adding stored procedures to the data objects which will enable automatic flattening of the composite data objects into various representations (e.g. text, XML, JSON, etc.). Furthermore, a mapping language will be defined for automatic translation of legacy data models such as relational into the hypergraph based data models, similar to our previous work presented in [14]. Formally, decomposition transforms the initial hypergraph $H = (\{v_i\}_{i \in I_V(H)}, \{e_j\}_{j \in I_E(H)})$ as follows:

$$H' = (\{v_i\} \cup \{v_l\}_{l \notin I_V(H)}, \{e_j\} \cup \{(\{v_s\}_{s \notin I_V(H)}, \{v_k\}_{k \in I_V(H)})\})$$

The difference is that composition adds a single vertex grouping a subgraph of objects while decomposition connects existing vertex to new vertices.

5 Representing Existing Data Structures and Formats in Filess

Typical data structures can be represented in hypergraphs in the following ways. Sets can be trivially created by creating a 1-N directed hyperedge. Lists can be created by linking consecutive nodes through a single hyperedge with identical ID such as "_:next", the actual property name is irrevelavant as long as the application wants to interpret a path as a list it is allowed to. However for performance reasons, a special type of node which allows to create order lists has been added. Maps are naturally represented by creating an hyperedge from a head node to any number of tail nodes.

JSON is a text format used to represent key-value pairs, where keys are always strings, and values can be any of the following types: Number, String, Boolean, Array, Object and `null`. These types map almost naturally into Filess data model. Boolean values can be modelled using Number data object type,

Array's by creating lists and `null` values can be achieved using hyperedges with empty head sets. Object values can be directly represented using Composite data objects. One issue is that of namespaces, as the edges created from the JSON key's must be attached to some namespace in order to disambiguate them from other edges. By default JSON has no concept of namespace, so it is up to the application to provide one.

XML (eXtensible Markup Language) is a W3C recommendation which is a tree based model for representing structure data on the Internet. In contrast to JSON, it provides means for specifying unique namespaces for all elements, ordering of the nodes as well as assigning attributes to nodes (unordered). The mapping of XML data into directed hypergraph can be achieved as follows. All simple tags (containing only values) are converted to simple data objects. All complex tags, which contain children tags are converted to composite data objects. All tag attributes are added to respective data objects using edges.

The representation of relational model using directed hypergraphs can be achieved as follows, assuming that the database is at least in the 3rd normal form. Each relation is composed of a set of value tuples, called rows. Each row is simply mapped to a single composite data object with edges representing the columns and their particular values as target data objects. Each relation (i.e. table) can be represented as a set of data objects representing rows. More interesting is the case of foreign key dependencies. In case of relational model it is impossible to directly create *n:m* relations. Consider the relations *Author* and *Book*, where it is possible that a single book could have many authors as well as a single author could have published several books. In the relational model this requires introduction of intermediate relation (e.g. *BookAuthors*), which assigns authors to books. In case of a hypergraph this relation is not necessary (i.e. it is not necessary to create a new data object), as the relevant property can be modelled directly using hyperedges.

6 Prototype Design and Implementation

Filess provides users with an abstract API enabling basic operations on the data objects such as searching, creating and opening. As mentioned above, each user sees the global data space from their own perspective, which is identical on all devices from which they access the system. Current Filess prototype is implemented as an intermediate layer between user applications and distributed graph database backend (see Fig. 3). The current prototype implementation is created in Java language. *libfiless* library provides methods which can be used directly by users application. The Filess API provides lower level system calls which provide abstraction over currently used data storage backends, so that users applications do not get locked in into particular solution. For the prototype it also keeps the authentication sessions information in memory, however this will be in the future migrated to external database. The Filess layer provides an API with the following groups of operations:

- *Session* - These operation enable the user to login and logout of the system. Each session combines the users key with current machine ID so that the same user can be logged in from multiple devices simultaneously, and see the same state of affairs from these devices,
- *Get* - This category of operations allows for searching and access data objects. Currently the search is limited to node GUI's, as the Filess layer aims to be agnostic of actual graph database backend, an ongoing work is to develop an abstract query language for this purpose,
- *Put* - These operations enable adding new data objects and relating them to other objects,
- *Join* - These operations enable composing existing objects into more complex objects,
- *Split* - These operations enable decomposing existing binary or text objects into graph form,

In order to enable evaluation of the idea, Filess prototype has been developed using available technologies in the area of graph databases. We have evaluated several solutions including [11,15]. Finally we chose OrientDB, which is a multi-document database enabling modeling using document, key-value as well as graph paradigms simultaneously. In order to support legacy applications, an intermediate FUSE filesystem plugin was implemented which allows applications to access the information in the form of files which are composed on demand from the underlying graph when applications try to gain access to the data object. The implementation is based on *fuse-jna* Java Fuse provider, which allowed us to use direct OrientDB Java bindings. Due to very flexible graph model in OrientDB, it was possible to create hypergraph structure by defining custom edge class. Binary, read-only data objects are stored in a separate distributed database called IPFS (Interplanetary File System), which provides efficient hashing and distribution of large binary files between multiple nodes by diving them into blocks and maintaining a tree structure based on the blocks hash values.

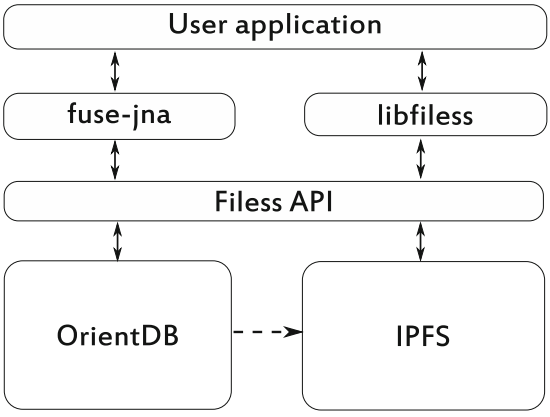


Fig. 3. Architecture of the Filess prototype

7 Conclusions

In this paper we have presented a novel approach to data management and representation in information systems, which departs from the filesystem based designs. Filesystem approach has become already intractable for average users for several reasons such as difficult searching for required files or lack of OS level synchronization of data between devices used to access the system. Presented approach addressing these problems has the potential to enable much more natural access to information, while minimizing the redundancy and data transfer on a global scale, allowing at the same time for highly fine grained access control, not based on files, but on actual data elements, which will enable creation of much more sophisticated and natural computing infrastructures able to handle information processing tasks on a global scale. The presented approach requires both users and application developers to shift the paradigm in which the applications are developed and used. Future work will include design of security layer enabling fine grained control over the operations performed by various users on such global data model, practical evaluation of performance depending on underlying storage solution and development of minimum viable prototype of the truly file-less operating system.

Acknowledgment. This research has been funded by Polish National Science Centre grant *File-less architecture of large scale distributed information systems* number: DEC-2012/05/N/ST6/03463.

References

1. Bandulet, C.: Object-based storage devices (2007). http://developers.sun.com/solaris/articles/osd.html
2. Berge, C.: Hypergraphs: Combinatorics of Finite Sets. North-Holland Mathematical Library, vol. 45. North-Holland, Amsterdam (1989)
3. Boyd, M., Mçbrien, P.: Comparing and transforming between data models via an intermediate hypergraph data model. In: Spaccapietra, S. (ed.) Journal on Data Semantics IV. LNCS, vol. 3730, pp. 69–109. Springer, Heidelberg (2005)
4. Drago, I., Mellia, M., Munafo, M.M., Sperotto, A., Sadre, R., Pras, A.: Inside dropbox: understanding personal cloud storage services. In: Proceedings of the 2012 ACM Conference on Internet Measurement Conference, IMC 2012, pp. 481–494. ACM, New York (2012)
5. Ferré, S., Ridoux, O.: A file system based on concept analysis. In: Palamidessi, C., Moniz Pereira, L., Lloyd, J.W., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Sagiv, Y., Stuckey, P.J. (eds.) CL 2000. LNCS (LNAI), vol. 1861, pp. 1033–1047. Springer, Heidelberg (2000)
6. Gallo, G., Longo, G., Pallottino, S., Nguyen, S.: Directed hypergraphs and applications. Discrete Appl. Math. **42**(2–3), 177–201 (1993)
7. Gantz, J., Reinsel, D.: The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. International Data Corporation, December 2010. https://www.emc.com/collateral/analyst-reports/idc-digital-universe-united-states.pdf

8. Gifford, D.K., Jouvelot, P., Sheldon, M.A., O'Toole, Jr., J.W.: Semantic file systems. SIGOPS Oper. Syst. Rev. **25**(5), 16–25 (1991)
9. Grimes, R.: Code name WinFS: Revolutionary file storage system lets users search and manage files based on content. MSDN Magazine 19(1) (2004). http://msdn.microsoft.com/msdnmag/issues/04/01/WinFS/
10. IDC iView: The Digital Universe Decade - Are You Ready? International Data Corporation, Framingham, MA, USA (2010). http://www.emc.com/digital_universe
11. Iordanov, B.: HyperGraphDB: a generalized graph database. In: Shen, H.T., Pei, J., Özsu, M.T., Zou, L., Lu, J., Ling, T.-W., Yu, G., Zhuang, Y., Shao, J. (eds.) WAIM 2010. LNCS, vol. 6185, pp. 25–36. Springer, Heidelberg (2010)
12. Kryza, B., Kitowski, J.: Comparison of information representation formalisms for scalable file agnostic information infrastructures. Comput. Inf. **34**, 473–494 (2015)
13. Kryza, B., Kitowski, J.: Filess - file-less architecture for future information systems. In: 2014 IEEE Fourth International Conference on Big Data and Cloud Computing, BDCloud 2014, Sydney, Australia, 3–5 December 2014, pp. 281–282 (2014)
14. Mylka, A., Mylka, A., Kryza, B., Kitowski, J.: Integration of heterogeneous data sources in an ontological knowledge base. Comput. Inf. **31**(1), 189–223 (2012)
15. Orient Technologies: OrientDB project website. http://www.orientechnologies.com
16. Padioleau, Y., Ridoux, O.: A logic file system. In: Proceedings of the General Track: 2003 USENIX Annual Technical Conference, San Antonio, Texas, USA, 9–14 June 2003, pp. 99–112 (2003)
17. Rajimwale, A., Prabhakaran, V., Davis, J.D.: Block management in solid-state devices. In: Proceedings of the 2009 Conference on USENIX Annual Technical Conference, USENIX 2009, pp. 21–21. USENIX Association, Berkeley (2009)
18. Reiser, H.: Futurue vision of reiserfs (2006). https://reiser4.wiki.kernel.org/index.php/Future_Vision
19. Salton, G.: Another look at automatic text-retrieval systems. Commun. ACM **29**(7), 648–656 (1986). http://doi.acm.org/10.1145/6138.6149
20. SanDisk: Ulltradimm product page. http://www.sandisk.com/enterprise/ulltradimm-ssd/
21. Seagate Technology LLC: The seagate kinetic open storage vision. Seagate Technology LLC (2013). http://www.seagate.com/tech-insights/kinetic-vision-how-seagate-new-developertools-meets-the-needs-of-cloud-storage-platforms-master-ti/
22. Stender, J., Hogqvist, M., Kolbeck, B.: Loosely time-synchronized snapshots in object-based file systems. In: IPCCC, pp. 188–197. IEEE (2010)
23. Wang, F., Brandt, S.A., Miller, E.L., Long, D.D.E.: OBFS: a file system for object-based storage devices. In: Proceedings of the 21st IEEE/12TH NASA Goddard Conference on Mass Storage Systems and Technologies, College Park, MD, pp. 283–300 (2004)
24. Williams, R.: How we found the missing memristor. IEEE Spectr. **45**(12), 28–35 (2008)