Richer File System Metadata Using Links and Attributes

Alexander Ames
sasha@cs.ucsc.edu

Nikhil Bobb
nikhil@cs.ucsc.edu

Scott A. Brandt
sbrandt@cs.ucsc.edu

Adam Hiatt
adamh@cs.ucsc.edu

Carlos Maltzahn
carlosm@cs.ucsc.edu

Ethan L. Miller
elm@cs.ucsc.edu

Alisa Neeman
aneeman@cs.ucsc.edu

Deepa Tuteja
deepa@cs.ucsc.edu

Storage Systems Research Center
Jack Baskin School of Engineering
University of California, Santa Cruz

Abstract

Traditional file systems provide a weak and inadequate structure for meaningful representations of file interrelationships and other context-providing metadata. Existing designs, which store additional file-oriented metadata either in a database, on disk, or both are limited by the technologies upon which they depend. Moreover, they do not provide for user-defined relationships among files. To address these issues, we created the Linking File System (LiFS), a file system design in which files may have both arbitrary user- or application-specified attributes, and attributed links between files. In order to assure performance when accessing links and attributes, the system is designed to store metadata in non-volatile memory. This paper discusses several use cases that take advantage of this approach and describes the user-space prototype we developed to test the concepts presented.

1. Introduction

Traditional file systems provide a weak and inadequate structure for meaningful representations of file interrelationships and other context-providing metadata. Solving this problem has become increasingly urgent as users are faced with a growing amount of personal data such as email, chat communications, digital photography, and online music. Moreover, computational scientists continue to bemoan the lack of mechanisms for cross-archival access, retrofitting of metadata, and identifying groupings of related results needed for data mining [38, 39]. A content-neutral, file system-based mechanism for storage of arbitrary metadata provides one solution to this weakness. As an application of this principle, we introduce the Linking File System (LiFS). It extends file system metadata to include not only arbitrary, user-specifiable key-value pairs on files but relationships between files in form of links with attributes.

It is now often easier to find a document on the Web amongst billions of documents than on a local file system. Documents on the Web are embedded in a rich hyperlink structure while files typically are not. Companies such as Google are able to take advantage of links between Web documents in order to deliver meaningful ranking of search results using algorithms like PageRank [32]. In contrast, search tools for traditional file systems do not have information about inter-file relationships other than the hierarchical directory structure and ownership of files.

The reason for this dearth of relationships between files is that the management of file system metadata is expensive in traditional system architectures where volatile main memory must be backed by much slower disk-based secondary storage. The advent of new non-volatile main memory technologies such as MRAM [8] promises to reduce the cost of accessing file systems in an arbitrary or fine-grain fashion with the development of novel file system designs [16, 27, 49].

The promise of non-volatile main memory has prompted designers to use such memory for the persistent storage of file system metadata. Although memory-resident metadata trivially speeds up certain common file system operations (such as `stat`), a more remarkable benefit is the ability to employ far richer metadata structures. File systems designers will no longer be constrained by disk access speed, but instead can focus on the needs of the user. Frequent access, context-aware searches, and other operations that are prohibitively expensive under traditional, disk-oriented file system architectures will become feasible.

The Linking File System's ability to assign attributes to and establish attributed links between files in a standardized fashion forms a powerful infrastructure capable of sup-

This paper was published in the *Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005)*, Monterey, CA, April 2005.

porting a variety of different and extremely useful user, application, and system operations. Attributed files directly support enhanced file system searches. Attributed links will support a number of recent efforts to extend hierarchical directory structures with more user-friendly and personalized file organizations [18, 28, 31, 35, 43]. Weighted links between files can also be used to record access patterns that are useful for pre-fetching, hoarding, indexing, and search result ranking. Indeed, these links provide an abstract model for file interrelationships previously unavailable at a file system level.

2. The Design of LiFS

The key features of LiFS are links between files and attributes on both files and links. To ensure the performance and reliability of LiFS, the design relies on both the non-volatility and low latency of MRAM. At the most basic level, a search within the file system traverses a series of links across a graph of metadata. An in-memory structure is crucial for this operation; random seeks on disk to inodes, even with some caching, would be prohibitively slow. The low latency of MRAM will allow metadata operations and searches to be performed almost instantaneously.

2.1. Links

Each link in LiFS has a source file, a target file, and a non-empty set of attributes consisting of key-value pairs. LiFS links differ from POSIX links in that LiFS links represent a relationship between files instead of simply a reference to a file. The attributes of the links express the nature of the relationship.

Any file can potentially contain a link to any other file. As a consequence, every file is also a directory and the distinction between files and directories is eliminated. The traditional notion of containment of a file within a directory is simply one relationship among many that can be expressed with links.

The key benefit of links is that they provide native support for a variety of relationships between files that are currently supported in an *ad hoc* manner by individual applications and the operating system. In addition to containment, links can express a variety of other useful relationships such as included-in, referenced-by, dependent-upon, created-by, opened-by, and others. Links also allow for dynamically customizable views of the file system based on the type of link followed.

2.2. Attributes

In our initial design both files and links can carry a number of attributes limited only by available memory. The size

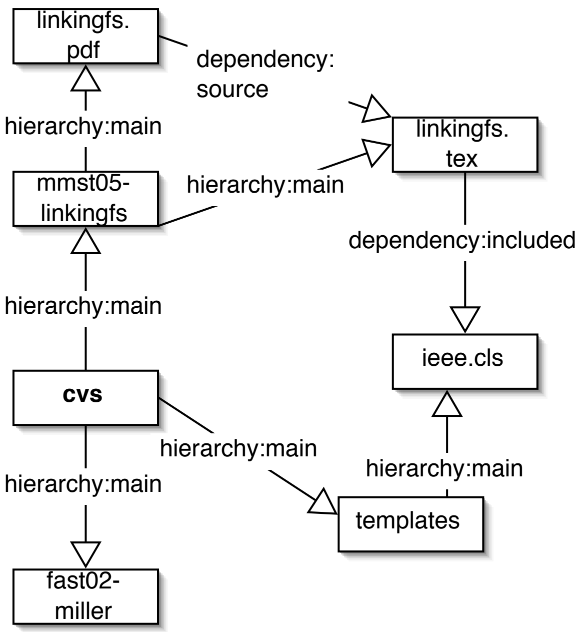

Figure 1. Example use of links with attributes: organizing files into a directory-like hierarchy called "main" and keeping track of dependencies between files.

of each key and its corresponding value is also currently unconstrained. We anticipate putting an efficiency ceiling on this size into the final implementation; excessively long key and value strings might affect the ability to deliver high performance.

Both the key and value members of an attribute can contain arbitrary data, including binary data. This allows applications to have rich metadata such as thumbnails, preview video clips, and cached printer spool files without the overhead of special encoding. Sharing of metadata via file and link attributes provides a powerful infrastructure for application integration.

The main benefit of attributes is that they enable users, applications, and the system itself to annotate files and links. This allows for fast and effective file searching, categorizing, partitioning, and manipulation, and provides infrastructure for other features that may not have been consider by the file system designer. It also provides a far richer context in which files can include information about provenance, intended use, type, contents, creator, modification history, version, and other information that a user, application, or system may want to keep.

We also allow attributes to be executable. A special case of executable file attributes are *file triggers*. A file trigger on a file specifies a pattern/action pair. A pattern specifies the file system operation (such as a read or write) on the file with which the executable attribute is associated. An action specifies code that is executed whenever the associated operation is invoked on the file. File triggers are a powerful mechanism that simplifies the implementation of a wide variety of file system services such as versioning, mirroring, and others (as discussed in Section 3.5). File triggers raise a rich set of interesting security and language design issues that we are actively investigating.

2.3. Interface

LiFS supports an enhanced version of the POSIX interface. Here, we present the key examples of the linking API.

2.3.1. Link Creation We introduce a new system call, `rellink`, to create new (relational) links. Creating links requires the specification of a source file, a destination file, and one or more attributes. The inclusion of attributes necessitates our new system call; otherwise standard POSIX semantics would be sufficient. For example, `mkdir`, where the source is an existing directory and the target is the new directory to be created, cannot be used because the `mkdir` call does not support attributes. Of course, `mkdir` can be implemented as a special case of `rellink` with the source implicitly specified and the attributes set to reasonable defaults. `rellink` creates a *hard* link from the inode of the source file to the destination file. The paths to source and target must identify each object, as opposed to using the inode numbers directly. We provide a similar call, `relsymlink`, that creates a soft link. This symbolic link is much like the POSIX `symlink` system call, but adds the assignment of attributes on the link itself, like its hard link counterpart. This allows the user to create relationships with objects and names in addition to creating relational links directly between objects,

2.3.2. Multiple Links and Identification In order to provide the greatest flexibility for relationships, multiple links are permitted between the same source and target. For example, two different users could each define their own relationships between the same pairs of files. However, to prevent confusion, each link must be *uniquely identifiable* by its attributes. In practice, this constraint is easily met since we expect that multiple links between files will usually belong to different users and will therefore at a minimum have different user attribute values.

Generally, a user may use LiFS like most conventional file system representations, where paths containing *names* identify files. However, in LiFS, name may be just one of many attributes given on the links between files. Therefore, it is possible to have paths to files using some attribute other than the conventional name. For example, we may have links between documents representing citations, *i. e.* a source document contains a link to a target when the source cites the target. This allows finding documents by specifying "citation paths."

Nonetheless, in order for users to properly identify files that they may wish to open, there must be a traversal of identifiable links beginning with a file system entry point to the target file, akin to root, or a path relative to the current working directory. Unlike conventional file systems, there need not be the same root directory for all users; in fact, a user may be able to choose between one of many roots.

2.3.3. Link Accesses, Updates and Deletions In LiFS, we introduce API calls to retrieve a single link, perform updates to the link, and delete a link. Just as users must open a single file in conventional systems, there are cases where one needs to open a single link, such as to retrieve additional metadata about the relationship expressed in the link; this is done with the `openlinkattrset` call, which retrieves attributes for a specified link. Users must uniquely identify a link, as discussed above, with a source path, target path, and enough attributes to uniquely match the attribute set of the link to be retrieved. The call returns an error if the specification does not match exactly one link; matching zero links is an error, as is matching more than one link. Similarly, the call `setlinkattr` must uniquely identify the link whose metadata one wishes to modify. New attributes may be added, or the values of existing ones may be modified where an attribute with the same key already exists. An overriding constraint on this function is that changes must not prevent other links from being uniquely identified after the change is complete. Finally, uniquely identified links may be deleted via an API call.

2.3.4. Link and Attribute Sets In addition to supporting standard POSIX operations and semantics, LiFS includes syntax for attribute-based searching and selection. For searching, we introduce two functions for fetching sets of links. The first, `openlinkset` allows the user to see all links outgoing from a specific source. The second, `openmatchlinkset` returns the links from a source that have matching attributes. For this call we specify attributes simply as a comma-delimited string. Our model for these calls are the POSIX calls such as `opendir`, `readdir`, etc. Likewise, LiFS contains additional calls `readlinkset` and `closelinkset`.

To provide seamless POSIX compatibility, `mkdir` creates a link with a distinguishing default attribute, `POSIX=TRUE`. Similarly, `open` on a new file creates a link from the current working directory to the file with attribute `POSIX=TRUE`. Thus, we can search a directory for a file created by `open` using either `opendir` *or* `openmatchlinkset`.

Attribute sets work in a similar fashion. The `openlinkattrset` call retrieves the attributes for a single link. When multiple links are retrieved with the calls introduced above, the link sets contain references to an attribute set for each link. We also have the calls `readattrset` and `closeattrset`. Once read, the attribute set structures have separate key and value string components.

2.3.5. Querying and Link Traversal To take full advantage of attributes and links LiFS will provide a query language that is more powerful than the POSIX interface. We are currently experimenting with a declarative query language that supports graph pattern matching, attribute value constraints, and optional values.

We build the link traversal graph using several smaller components and data structures. LiFS has three hash tables; one containing the link set for each inode, one containing file attributes, and one containing link attributes.

At the most basic level, each file and directory has a unique inode number, and each link has a unique *l_node*. We keep attribute sets for each file in linked lists. The lists can be queried from the `ino_hash` hash table, where each attribute list is hashed under the inode number of the file it describes. Each relational link's attributes are similarly contained in a linked list and stored in a second hash table under the relational link's l_node number.

Each l_node contains an l_node number, and a name and inode number of the file at which the link is pointing. The l_nodes for each file are contained in a linked list and stored in a hash table by the *inode number* of the file.

To traverse the graph, we simply need a file or directory in the graph from which to start. Using the inode number or name, we can consult the link set hash table for outgoing links, and filter the resulting target inode numbers by attribute (either file attribute or link attribute). From there, we can continue outward in a pruned breadth-first traversal for as many hops as desired. These data structures are much more efficient than a database, since the high overhead of a join operation is avoided. Additionally, the non-volatility of MRAM provides an ideal environment for using linked data structures.

In order to minimize the metadata footprint, we store names and values in a string pool which holds only one copy of each unique binary value. This also facilitates fast comparison since we need to perform a byte-by-byte comparison only when we add or change an attribute key or value. For other operations, we may instead compare string table indices. Additionally, we propose to use compression techniques, similar to those used by Edel, *et al.* [16] to most efficiently use MRAM space.

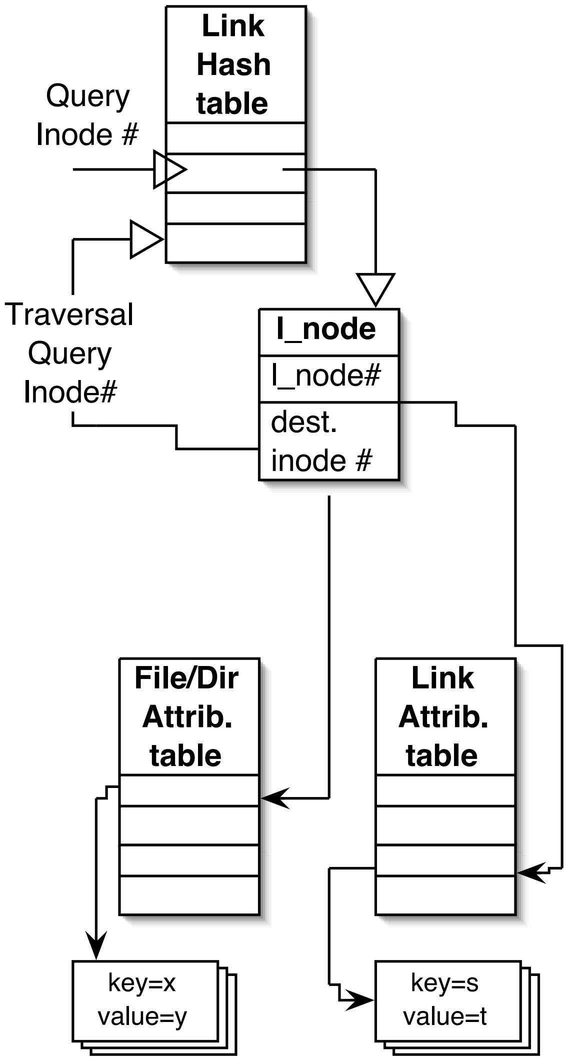

Figure 2. Hash tables to support links and attributes. A query returns links which can by filtered by attribute and can also seed a deeper traversal.

3. Use Cases for LiFS

The attributes and attributed links of LiFS enable a wide variety of functionality. In this section we provide a use case taxonomy that illustrates how users and applications can take advantage of the features of LiFS.

3.1. Infrastructure Dependencies

Any use of a file requires a certain infrastructure. For example, while ASCII-encoded text files can be displayed by a large number of applications, the display of more complex media such as movies or music is often dependent on a particular version of a particular application with a particular set of plugins. The same is true for executing programs, which may depend on having the correct versions of many dynamically-loaded libraries. Infrastructures to support such software can be complex, even on a personal computer, since a properly functioning system can often depend on hundreds of applications and operating system services. Each installation or upgrade can break some of these dependencies if not handled properly [4]. The following are examples of how LiFS can help to tame this complexity.

3.1.1. Multiple Concurrent Libraries Correctly configuring library dependencies in Linux and BSD environments is often complex and time consuming [4]. LiFS allows easy library dependency management by linking an application to all of its dependent libraries via a dependency attribute on each link. When starting an application a shell creates a custom environment in which to run the application from the information embedded in the file system.

3.1.2. Multiple Platform Support Attributed links simplify the support of multiple platforms within one file system. Installations for multiple platforms link platform-specific files to a common file system using links with a platform attribute. This approach has the advantage over the fat binary approach in that applying a platform-specific update does not require recompilation of all platforms. In addition, files that are particular to a subset of the platforms are only visible on those platforms for which they are relevant. This can include both machine-specific files, such as host name information, as well as platform-specific files, such as libraries or processor family-specific configuration information.

3.1.3. Infrastructure Change Alerts When removing or upgrading some infrastructure component such as hardware, operating system modules, or application software, infrastructure dependencies of digital objects might break, resulting in an infrastructure that has lost the ability to correctly run programs or adequately display digital objects. LiFS will not only be able to alert the user of infrastructure dependencies but also list the files affected by the change. These dependencies can also be a useful basis for deciding on what data to backup.

Changes in infrastructure are the primary source of obsolescence that creeps into digital data over time. By the time the user discovers the inability to display a file or execute a program properly it can be difficult to fix the infrastructure due to unrecoverable software or hardware. By maintaining dependency links to files necessary to interpret a given file, the system can notify a user when a needed file is about to be removed or changed.

3.2. Views and Contexts

Relationships between files and file attributes allow the specification of views and contexts. A file system view is a selection of the overall file system in terms of certain values of file and link attributes. A context refers to how a particular file relates to other files.

3.2.1. Customizable Views of the File System and Jails LiFS allows users to have customizable views of the file system. This is achieved by linking a user's personal root directory, either directly or indirectly, to all desired files across the file system. Once this view exists, the user is free to install applications and libraries anywhere on the file system instead of just being confined to personal directories. Moreover, a user can add his/her own links between two files, and search for files using personal links first and "default" system links second.

Such customizable views allow partitioning of the file system in a fashion similar to that offered by the file system-specific portions of BSD jails [21]. This functionality could either be offered directly by overlaying a separate file system hierarchy for the user in the jail, or indirectly by instructing the system to only allow users to access and follow links with an in-jail attribute particular to that user or their group.

Views can be customized even further according to the semantics of the attributes. For example, with a conference paper currently open in the file system, a user can create a view in which all referenced papers and figures are ordered according to the authors, the levels of reference to the current paper, and so on. To retrieve information about one of the authors, another view can present the files associated with the author such as pictures, contact information, and papers written by him/her.

3.2.2. Personal Context The use of directories in traditional file systems and the use of brittle symbolic links to circumvent limitations of a strict hierarchical structure has proven woefully inadequate for personal information management. Recently, a number of personal information system applications have been developed that attempt to fix the lack of relationships between files on the application level [30, 34]. We believe that the lack of relationship between files is not an issue limited to personal information management applications but rather a more general problem of data management and should therefore be solved in the file system layer. Recent announcements by Microsoft (WinFS) and Apple [2] (Spotlight) confirm this notion.

3.3. Provenance and History

All files in a file system have a history. The beginning of a file's history is its provenance. As files are manipulated and moved history information accumulates automatically. Provenance and history provide metadata that can be very useful for finding and organizing information. A file can be a result of computation, extracted from personal communication, created locally, or downloaded from the Web. Knowing the details of a file's history can be useful in many areas, including caching (files can often be refetched from the Web), volatility and permanence, and intrusion tracking.

3.3.1. Computation In a strict sense, all files are results of computation. However, files originating from a software build or a scientific calculation can often be regenerated, while data created by a user cannot be recalculated. The huge data sets spread across many files that are generated and examined in scientific computing, in particular, can benefit from relationships that can be expressed using links with attributes. Links are also useful for incrementally rebuilding complex systems in the presence of frequent changes.

Springmeyer, *et al.* note that scientists at Lawrence Livermore need better means for exploring and mining scientific results [39]. LiFS can tie together a raw data set by providing a user-customizable arbitrary out-of-core data structure on a collection of files. For example, a cluster based simulation may generate 16 files for a single time step, multiplied by 100 time steps. Each file could have links to its immediate neighbors within a single time step, and to the same location in previous and following time steps, essentially creating a 4-dimensional "megafile." Linked data sets can be particularly useful for data mining techniques such as region growing or for isosurface extraction: once a file with a particular data characteristic has been found, its immediate neighbors can be easily explored for the same feature. Additionally, linked groups of data sets can provide hints for other areas of exploration. By providing the linking in the file system, LiFS enables multiple applications to share the infrastructure, much in the same way that files and pipes in UNIX allowed multiple text-based utilities to cooperate.

LiFS also allows compilers, editors, and other development tools to software development information in the file system. Currently, product-specific files can contain development information such as compile and linking dependencies. UNIX `make` specifications offer some level of interoperability between environments, but are not perfectly flexible. As an alternative and potentially better solution, LiFS can store dependency information in attributes of links.

3.3.2. Personal Communication Email or instant messaging can deliver attachments, calendar entries, "to-do" items, and other files to users. Recording the provenance of these attachments in file attributes or file links automatically creates a useful record of the context in which these files were acquired. Email and chats themselves can be explicitly related to files representing correspondence partners.

3.3.3. Web Context As soon as a web resource is downloaded to a traditional file system it typically loses its Web context, though programs such as DownloadComment on the Macintosh can record source URLs as a file comment [15]. This loss of context is a major reason that it is often easier to find files on the web than in the local file system. A file's provenance URL allows a file system search engine to tap into data available at web search engines in order to rank files and to determine relationships between downloaded files. With Web context information one can determine whether a new version of a file is available or whether the corresponding Web resource disappeared. The latter case is important for decisions on how to digitally preserve local copies.

3.4. Searching

Methods such as Google's PageRank [32] show that rich relationships between web resources can help to generate useful ranking of search results. As mentioned in earlier, search in file systems is handicapped by the low number of inter-file relationships. We believe that relationships introduced for the use cases mentioned above will provide a wealth of information to make search result ranking more relevant. For example, consider a scenario where prioritized "to-do" items are represented as files. These items refer to projects that in turn link to other files. A user searching for the files that need to be dealt with most immediately could quickly identify the files associated with the most urgent action items. Even though the files themselves do not have a priority on them, the priority information on the action items could be propagated to the files by a search ranking algorithm.

3.5. Workflow and File System Services

The management of large collections of files frequently requires batch processing with some workflow complexity; examples include migration between physical storage media, migration between document formats, publishing, and backups. Workflow states and histories can easily be represented in file and link attributes instead of relying on a repository specific to a workflow management application. We expect that many workflow actions can be implemented by file triggers, discussed in Section 2.2.

When correctly used, file triggers can provide a powerful infrastructure that simplifies the implementation of a number of workflow services, including those that are often thought of as file system services, such as mirroring, versioning, notification, data protection, digital rights management, and defragmentation.

Mirroring, copy-on-write, and snapshots can be implemented so that each write either redirects the write or causes an additional write. Versioning can be implemented by having executable attributes convert basic writes into differential writes. Code in executable attributes can control the degree of synchronization in mirrors or the granularity of versioning by using persistent memory to aggregate changes to some desired level. Scripts in file triggers can record file system operations for accounting and notification purposes. Read and write filters can implement encryption and real time virus checking using file triggers that operate on read and write calls. File triggers can also be used to implement operations such as file level defragmentation, digital rights management, and dynamic permissions. For example, file triggers can delete digital rights-managed documents after a certain number of views or after having their viewing limited to a certain time of day. Thus, LiFS has the potential to facilitate copy protection on the file system level.

Different file system services will require different forms of propagation of file triggers. For example, for the implementation of snapshots propagation may need to occur immediately and globally while in other cases a lazy propagation-on-access strategy may be more appropriate.

4. Implementation

To test our design, we implemented a prototype of LiFS for Linux. This prototype runs in user space and can be mounted through an interface provided by FUSE (File System in User Space), a Linux kernel module [41] that redirects file system accesses to a user space program. This approach significantly reduced development time by allowing us to spend much less time on the intricacies of kernel programming.

The POSIX standard includes no specification for any entity comparable to the relational link and the development of the Linux virtual file system has found no need to provide such an extension. Therefore, we created user space applications to replace nonexistent native system calls. The mechanism to place attributes on links and files was implemented in a similar fashion.

We used a PostgreSQL database [50] to hold LiFS metadata. The database consisted three tables: one each for file inodes, links, and link attributes. Since links were identified by their source and destination inodes, identifying the links originating or terminating at a particular inode could be done by a simple lookup. The use of a separate table for link attributes allowed a simple design that permitted multiple attribute-value pairs per link. The use of a database simplified data storage and let us focus on defining functionality without worrying about the complexity inherent in the design of a space and speed conscious metadata store. Future plans include replacing the database with light weight data structures that are optimized for LiFS workloads. Additionally, we use the underlying Linux file system on disk as our store for file data.

For reasons of efficiency, dynamic link libraries (DLLs) were chosen for triggers. This allowed the file system to only have the minimal overhead of a function call in order to call a trigger. To limit complexity, each file may have only one trigger on each of its filesystem calls. Additionally, the assumption was made that a filesystem with triggers would have a proportional amount of MRAM to it's available disk storage.

Overall, the functionality of triggers was split into four parts: the file system hooks, the trigger controller, the interface for the individual modules, and the modules themselves. A module is defined as a group of triggers which works together towards a specific end; each module is stored in a DLL. The advantages of using a DLL include high performance and no system downtime while replacing modules or refreshing module data.

The trigger controller determines whether a trigger should be activated on a particular filesystem call. It maintains a list of currently activated modules, and a table which hashes the combination of system call and filename to a function pointer to the relevant trigger. This hash table is stored in main memory and is written through to simulated MRAM.

The operation of triggers can be illustrated using the example of trigger-based snapshots. The snapshot module can be controlled on a per file basis. Snapshots can be taken on any multiple of one second. To achieve snapshotting, the snapshot module has a trigger on the write call. This trigger checks if the set snapshotting frequency has elapsed between the last write and the current write, and if so mirrors the new call to MRAM, recording the time it was made. If the frequency has not elapsed, the current write is mirrored without recording the time it was made. This approach to snapshotting is advantageous in that it only has overhead when writes are actually being made, and is granular on the file level.

While our current implementation of LiFS is functional, it is still a prototype, and will require implementation using more efficient data structures to make it sufficiently fast for use in a real system. Challenges we face in accomplishing this goal are discussed in Section 6.

5. Related Work

The concepts we use in the design of LiFS borrow from various research areas ranging from semantic file systems to databases and the Web. We first look at file systems with queryable metadata, such as semantic file systems, and file systems designed to run in nonvolatile memory. We then touch upon upcoming advanced commercial systems and advances in active infrastructures. Finally we look at the Semantic Web and archiving, and how they try to convey knowledge more accurately and for the long term.

5.1. Semantic and Other Queryable File Systems

The Semantic File System [18] was originally designed to provide flexible associative access to files. File attributes, expressed as key-value pairs are extracted automatically with file type-specific transducers. A major feature of this work is the concept of virtual directories, in which a user makes an attribute-based query and the system creates a set of symbolic links to the files in the result set, providing access that crosses the directory hierarchy. A similar file system, Sedar [25], is a peer-to-peer archival file system with semantic retrieval. Sedar introduces the idea of semantic hashing to facilitate semantic searching and reduce storage and performance costs. A user may search for files semantically similar to a query file, and, like SFS, the user can request a directory to be created containing the results of a query.

The Inversion File System [29] uses a database to store both file data and metadata. The database also provides transaction protection, fine-grained time travel, and instantaneous crash recovery. Each file is identified by a unique ID, but also has a name and directory associated with it. A hierarchical namespace is imposed, but one can also query by file name and metadata. Moreover, Gupta, *et al.* cite the difficulty of managing different but related sets of files as motivation for their fan-out unification file system, in which fan-out unification refers to merging two directories, and implicitly, entries in a directory are treated as members of a set [19].

The Logic File System (LISFS) uses a database to support queries for sets of files in the system [31]. Database tables are composed of mappings from keywords to objects. Keywords look like directories, but keywords in a query can be in any order on a directory path, so there is not a strict hierarchy. The contents of a directory is the set of objects that meets the criteria of the relation in the query. The queries are very expressive; boolean expressions can be used to differentiate among files with the same name. Like the above mentioned file systems, the use of attributes in LiFS allows a user to perform expressive queries to locate files. However, these file systems all use secondary storage for metadata, either with or without a database, and must operate under such performance constraints. Additionally, none contain a linking mechanism that supports attributes, thus allowing inter-file relationships that can express the structure of the Web or establish data provenance and history.

5.2. In-Memory File Systems

In the HeRMES system, we suggested that MRAM be used to store file system metadata [27]. We noted that metadata overhead for a file system is 1–2 percent (or 600 MB of RAM for 60 GB of storage), and suggested the space requirements could be reduced using various compression schemes and algorithms. Edel, *et al.* demonstrated that inode storage space could be reduced by an order of magnitude (from 128 bytes per inode to between 15 and 20 bytes) [16] by stripping out unused fields and gamma-encoding others. They also found that there was no significant difference in performance compared to non-compressed metadata—the space saved by compressing metadata was nearly free.

Conquest [49], another non-volatile memory-based file system, utilizes persistent RAM for storage to alleviate disk traffic. In addition to metadata, Conquest stores all small files in persistent RAM. Unlike HeRMES and LiFS, which plan to utilize MRAM, Conquest has explored the use of battery-backed DRAM as its form of persistent RAM. Also, Conquest uses a traditional hierarchical file system, and lacks the advanced file system features facilitated by utilizing persistent RAM.

5.3. Advanced Commercial File Systems

WinFS is Microsoft's in-development file system, for which they have published a preliminary description of planned features[1]. WinFS appears to be a marriage of a database for metadata and NTFS for file stream performance, akin to our LiFS prototype usage of a database and the underlying Linux file system for data storage. Initial plans call for the file system to reside in a subdirectory of the "main" file system, rather than completely replacing the original file system. A key rationale for the file system is the ability to search and share attributes and data from diverse sources such as a user's address book, song files, and other material. The API is object-oriented; objects are called *Items*. Items cover a full range of granularity from simple descriptions, such as "author", to collections such as folders. The database backing allows SQL-type queries, XPATH searches [12], and additional power coming from

[1] This comes with the caveat that all is subject to change. Further information is available at `http://longhorn.msdn.microsoft.com/portal_nav.htm`

Microsoft's OPATH, a query language designed for a directed acyclic graph of objects [36].

Apple Computer's Spotlight technology will ship in 2005 with their new Tiger operating system [2]. Spotlight is a metadata and content indexing system that is integrated into the HFS+ file system. As with WinFS, metadata is stored in a database; Spotlight indexes file content and includes the results in the database as well. The system indexes a file when it is created, saved, moved, copied, or deleted. Spotlight provides *plugins* to extract content and attributes from well-known file types such as Word and PDF documents, images, movies, email, and contact lists. As with the Semantic File System's transducers, a user can write his/her own file type-specific plugin. Apple provides simple command line tools to query for files based on attributes and content, or to query the metadata description for a file. Apple's approach could benefit from a LiFS-like linking mechanism with metadata allowing relationships between content to be expressed; Spotlight currently only allows indexing on files, not the links between them.

Sun Microsystems has designed their file system, ZFS, for larger capacity, faster operation, and consistency checking, with reduced administration costs. Moreover, ZFS allows administrators to configure individual file systems for users or application, all allocated from a single pool of storage [1]. Like ZFS, LiFS allows for the definition of unique and dynamic file systems per user or per application by virtue of links with attributes. Thus, it is possible to create file systems on demand utilizing either system. However, ZFS does not contain the rich metadata constructs present in LiFS.

5.4. Active Infrastructures

There have been several document management systems that have embodied concepts similar to those used in LiFS. Dourish, *et al.*'s active document management system called Placeless Documents includes both passive name/value properties and active properties attached to documents [14]. Each active property consists of a name and an executable Java object. Sample activities include automatic backup, versioning, and consistency checking of document copies. Execution of active properties can be triggered by requests such as read, write, or delete in a way similar to LiFS file triggers. Dourish, *et al.* note that the composition of active properties can create significant complexities since a single request can trigger multiple actions. The issue of composition of active properties on documents is very similar to that of file triggers, and a similar approach can be taken to simplify the issue of composition. Placeless Documents uses a two-pronged approach in which properties are separated into three phases and are then explicitly ordered. The three phases are the verify, perform, and notify phase. Properties run during the verify phase are those which are involved in deciding whether a specific operation is allowed, such as a property granting access permission. Properties run during the perform phase are those that carry out an operation itself such as a compressed read. Lastly, properties run during the notify phase are those which are involved with clean-up after an operation or notification upon an operation's completion. This two-pronged approach is directly applicable to making the composition of file triggers more manageable. LiFS' file triggers are a generalization of active properties in that they are integrated in a file system as opposed to a document management system.

5.5. The Semantic Web

The original World Wide Web has expanded upon the ability of traditional documents to convey knowledge by adding links. Traditional references in documents, if given at all, would require a reader to manually find the "referred to" document. However, within the World Wide Web and hypertext documents in general, links allow readers to automatically traverse from one document to another when the document refers to the other. The Semantic Web expands upon this by allowing the links themselves to contain information about that particular relationship from one document to another. On top of that basic framework, one may devise ontologies to further convey knowledge in ways not possible through previous means. [6].

To make the Semantic Web possible, authors at the W3C have been developing various standards for its implementations in a way analogous to the standardized HTML and HTTP for the World Wide Web. The group of Semantic Web standards fall into layers, with URI and unicode on the bottom, XML, name spaces, and schemas to comprise the self-descriptive document layer in the middle, and the RDF layer on top. The function of this layer is to provide a common framework for metadata across applications. Atop the three bottom layers are additional layers for ontology vocabularies, logic, proof, and trust [5, 22]. The ontology layer has room for different attempts to devise languages in which to describe ontologies, such as OIL [17].

We envision three potential expectations for the Semantic Web. For humans, it may be a readily accessible universal library. Moreover, and in line with the ideas of its inventors, the Semantic Web has increased potential for machine processing of its contents, and this introduces the other two perspectives: the knowledge navigator and the federated knowledge or database [26].

Whereas the Semantic Web allows for the addition of richer metadata for the World Wide Web, it does so on a global scale. LiFS allows for the same depth of knowledge representation through its links and attributes. It accomplishes this within the scale of the local file system, which to many users, contains the data oh which the semantics of relationships matter most. Additionally, the Semantic Web RDF format can be basically broken into tuples of a subject, property, and object. Links within LiFS likewise contain source, attributes and a target. Thus, we can express the same relationships locally that are possible given the richness of the Semantic Web. Based on their similarities, LiFS could make an excellent file system or storage layer for Semantic Web data.

5.6. Digital Preservation

Digital objects do not survive until one makes a conscious effort to preserve them. This is in contrast to artifacts such as books, papyrus, and cuneiform tablets which exist until someone or something actively destroys them. One reason for the ephemeral quality of digital media is unreliable physical media: the shelf life of magnetic tape, hard drives, and CD-Rs can be less than a decade [11, 33, 3]. However, the loss of data due to unreliable media pales in comparison to the loss of data due to the rapid obsolescence caused by technological change. The rate of obsolescence is exacerbated by the fact that viewing of digital objects relies upon a complex infrastructure of software that is itself subject to frequent changes. Moreover, digital objects are often related to other digital objects that might change names or disappear entirely [7]. There are now large national and international efforts to address these issues; these efforts aim to provide standards for an exhaustive list of aspects for digital preservation in museums and libraries [46, 23, 44].

Trusted digital repositories [37] adhering to the now dominant international standard of the OAIS Reference Model [13] are an important component of digital preservation. Examples of digital repositories are DSpace [42], a collaboration of MIT and HP, New Zealand's Greenstone Digital Library Software [20], and Fedora [40], which is developed by the University of Virginia and Cornell. Common challenges of these digital repositories are scalability, interoperability with other repositories, and efficient workflow support for the entry (also called *ingest*) of large numbers of digital objects.

All these digital repositories are designed for use on an institutional level. However, the combination of unreliable storage and obsolescence unintentionally destroys much of digital media long before it can be considered for digital libraries. Personal correspondence and images that survived from earlier times form a significant part of our cultural heritage [24]. Today, personal correspondence in the form of email and chats as well as personal photos and movies are largely kept on home computers that neither meet standards nor follow practices of national digital libraries and are therefore unlikely to survive. In the future our period will be referred to as the "digital dark ages," according to Stewart Brand [10] and the list of significant losses of invaluable digital data during the last thirty years is daunting [45].

The design of LiFS provides the infrastructure to make digital preservation an integral part of file systems. Links and attributes can be used to explicitly represent the dependencies of digital objects on the software infrastructure thereby preventing accidental obsolescence or at least alert users to obsolescence events introduced by a particular change in the software infrastructure. Our hope is that this will make it easier for users to maintain good digital preservation practices so that at least the data that users want to preserve actually survives.

6. Conclusions and Future Research

LiFS is currently in an initial prototype state. However, the design and implementation work has already produced a number of new concepts and interesting challenges that are very encouraging. As we illustrated in Section 3, maintaining relational links and attributes in file system metadata turns out to be useful. Provenance links and attributes can embed local files into either the Web's hyperlink structure or threads of email or chats allowing for a new level of integrated search for Web resources, personal communication, and local files. Links are also a useful mechanism for prefetching, hoarding, and virtualization. File attributes provide a simple but powerful infrastructure for application integration. File triggers allow for a straightforward implementation of a variety of useful file system services.

Our initial work suggests many avenues for future research. In the near term, we plan to investigate efficient graph data structures in place of a database. Other avenues for exploration include deployment on a distributed system, providing a means for seamless search and traversal across both local and distributed links. This opens up a further issue of metadata placement among nodes to balance load and avoid hot spots. We will explore how we can leverage outcomes of the Semantic Web standardization efforts [48] for this effort. For example, the design of the LiFS query language is inspired by early results of an RDF query language [47].

As an alternative to traditional indexed searches we are investigating the use of multi-point indices which are implemented via index indirection. Indices in such a system would be organized hierarchically starting from a root index which points further down the tree. On a file system with arbitrary views of its structure for each user, this approach would be more secure because it would limit a user's searching of other users' structures. Implementing such a level of security with traditional indices would require the index to carry access information or to cross-reference with the system, both of which slow down searching. Additionally, we are hoping multi-point indices will offer compelling efficiency improvements.

File triggers present many issues which are yet unresolved; among them are the issues of protection and composition. Out of the approaches we have investigated, the Open Kernel Environment (OKE) [9] seems a likely candidate to resolve the related issues. The OKE uses a combination of trust management, a customized version of C (OKE-Cyclone), and a specialized compiler to provide protection with minimal runtime cost. The protection provided includes control over resources like processing time, stack space, and access to kernel data structures. Ideally, there would be a system of rules which predetermines an upper bound on the resource requirements of a trigger, partially at compile time and partially at run time, and then this upper bound would be used by the OKE as a hard limit on the resource usage of the trigger.

The addition of links and attributes to the file system opens the possibility of exploring file relationships to improve many applications, including data mining and visualization. Beyond simply improving applications, however, LiFS has the potential to improve the user experience by making data easier to organize, find, and retrieve. LiFS provides a rich infrastructure of files with attributes connected by attributed links that can be used and shared by current programs as well as applications that are as yet unimagined.

Acknowledgments

This research was funded in part by National Science Foundation grant 0306650. Additional funding for the Storage Systems Research Center was provided by support from Engenio, Hewlett Packard Laboratories, Hitachi Global Storage Technologies, IBM Research, Intel, Microsoft Research, Network Appliance and Veritas.

References

[1] Anonymous. In a class by itself - the Solaris 10 operating system. Technical report, Sun Microsystems, Nov. 2004.

[2] Apple Developer Connection. Working with Spotlight. http://developer.apple.com/macosx/tiger/spotlight.html, 2004.

[3] Associated Press. CDs, DVDs not so immortal. In *CNN.com*, May 6 2004. last viewed on Jan 9, 2005 at http://www.longnow.org/10klibrary/darkarticles/ArtCDROT.htm.

[4] S. Beattie, S. Arnold, C. Cowan, P. Wagle, C. Wright, and A. Shostack. Timing the application of security patches for optimal uptime. In *USENIX 16th Systems Administration Conference (LISA 2002)*, Philadelphia, PA, Dec. 2002. USENIX.

[5] T. Berners-Lee. Semantic web roadmap, Sept. 1998. Available at http://www.w3.org/DesignIssues/Semantic.html.

[6] T. Berners-Lee and E. Miller. The semantic web lifts off. *ERCIM News*, 51, Oct. 2002.

[7] H. Besser. Digital longevity. In M. Sitts, editor, *Handbook for Digital Projects: A Management Tool for Preservaton and Access*, chapter 9, pages 164–176. Andover: Northeast Document Conservation Center, 2000.

[8] H. Boeve, C. Bruynseraede, J. Das, K. Dessein, G. Borghs, J. De Boeck, R. C. Sousa, L. V. Melo, and P. P. Freitas. Technology assessment for the implementation of magnetoresistive elements with semiconductor components in magnetic random access memory (MRAM) architectures. *IEEE Transactions on Magnetics*, 35(5):2820–2825, Sept. 1999.

[9] H. Bos and B. Samwel. Safe kernel programming in the OKE. In *Proceedings of OPENARCH'02*, New York, USA, 2002.

[10] S. Brand. Written on the wind. *Civilization Magazine*, 5(5), October/November 1998. available at http://web.archive.org/web/19991110120021/www.civmag.com/articles/C9811F04.html.

[11] F. R. Byers. Care and handling of CDs and DVDs: A guide for librarians and archivists. Report 121, Council on Library and Information Resources and National Institute of Standards and Technology, October 2003. Last viewed on Jan 5, 2005 at http://www.clir.org/pubs/reports/pub121/contents.html.

[12] J. Clark and S. DeRose. XML path language (xpath), 1999.

[13] Consultative Committee for Space Data Systems. Reference model for an open archival information system (oais). Standards Recommendation 650.0-B-1 (Bluebook, Issue 1), CCSDS, January 2002. This Recommendation has been adopted as ISO 14721:2003.

[14] P. Dourish, W. K. Edwards, J. Howell, A. LaMarca, J. Lamping, K. Petersen, M. Salisbury, D. Terry, and J. Thornton. A programming model for active documents. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST '00)*, pages 41–50, 2000.

[15] Ecamm Network. DownloadComment software for Mac OS X. http://www.ecamm.com/mac/free/, 2004.

[16] N. K. Edel, D. Tuteja, E. L. Mlller, and S. A. Brandt. MRAMFS: a compressing file system for non-volatile RAM. In *Proceedings of the 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '04)*, pages 596–603, Oct. 2004.

[17] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. Patel-Schneider. Oil: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–44, 2001.

[18] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole, Jr. Semantic file systems. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP '91)*, pages 16–25. ACM, Oct. 1991.

[19] P. Gupta, H. Krishnan, C. P. Wright, M. Zubair, J. Dave, and E. Zadok. Versatility and Unix Semantics in a Fan-Out Unification File System. Technical Report FSL-04-01, Com-

puter Science Department, Stony Brook University, January 2004.

[20] K. J. Jon, D. Bainbridge, and I. H. Witten. The design of greenstone 3: An agent based dynamic digital library. Technical report, Department of Computer Science, University of Waikato, Hamilton New Zealand, December 2002. last viewed on Jan 9, 2005 at http://www.sadl.uleth.ca/greenstone3/gs3design.pdf.

[21] P.-H. Kamp and R. N. Watson. Jails: Confining the omnipotent root. In *Proceedings of the 2nd International System Administration and Networking Conference (SANE 2000)*, Maastricht, Netherlands, May 2000.

[22] M.-R. Koivunen and E. Miller. W3C semantic web activity, Nov. 2001.

[23] Library of Congress. Library of Congress announces awards of $15 million to begin building a network of partners for digital preservation. http://www.digitalpreservation.gov/about/pr_093004.html, September, 30 2004.

[24] C. Lynch. The battle to define the future of the book in the digital world. *First Monday*, 6(6), June 2001. available at http://firstmonday.org/issues/issue6_6/lynch/index.html.

[25] M. Mahalingam, C. Tang, and Z. Xu. Towards a semantic, deep archival file system. Technical Report HPL-2002-199, HP Laboratories, Palo Alto, July 2002.

[26] C. C. Marshall and F. M. Shipman. Which semantic web? In *HYPERTEXT '03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, pages 57–66. ACM Press, 2003.

[27] E. L. Miller, S. A. Brandt, and D. D. E. Long. HeRMES: High-performance reliable MRAM-enabled storage. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pages 83–87, Schloss Elmau, Germany, May 2001.

[28] S. Nickell. GnomeStorage. http://www.gnome.org/˜seth/storage/index.html, October 2004.

[29] M. A. Olson. The design and implementation of the Inversion file system. In *Proceedings of the Winter 1993 USENIX Technical Conference*, pages 205–217, San Diego, California, USA, Jan. 1993.

[30] Open Source Application Foundation. What's compelling about Chandler: A current perspective. http://www.osafoundation.org/Chandler_Compelling_Vision.htm.

[31] Y. Padioleau and O. Ridoux. A logic file system. In *Proceedings of the 2003 USENIX Annual Technical Conference*, pages 99–112, San Antonio, TX, June 2003.

[32] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford, Nov. 1998.

[33] M. Pollitt. Ever decreasing circles. In *The Independent*, 21 April 2004. Last viewed on Jan 5, 2005 at http://www.fbia.org/print.asp?ID=27963.

[34] D. Quan, D. Huynh, and D. R. Karger. Haystack: A platform for authoring end user semantic web applications. In *2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, FL, USA, Oct 20-23 2003. Last viewed on Jan 10, 2005 at http://haystack.lcs.mit.edu/papers/iswc2003-haystack.pdf.

[35] H. Reiser. Future vision. http://www.namesys.com/whitepaper.html, October 2004.

[36] T. Rizzo and S. Grimaldi. Data access and storage developer center: An introduction to "WinFS" OPath, 2004.

[37] RLG. Trusted digital repositories. Report, RLG-OCLC, Mountain View, CA, May 2002. Last viewed on Jan 9, 2005 at http://www.rlg.org/en/page.php?Page_ID=583.

[38] D. Schade, P. Dowler, R. Zingle, D. Durand, S. Gaudet, N. Hill, S. Jaeger, and D. Bohlender. A data mining model for astronomy. In *Astronomical Data Analysis Software and Systems IX*, pages 3–30, San Francisco, CA, 2000.

[39] R. Springmeyer, N. Werner, and J. Long. Mining scientific data archives through metadata generation. In *First IEEE Metadata Conference*, Apr. 1996.

[40] T. Staples. The Fedora Project: An open-source digital object repository management system. *D-Lib Magazine*, 9(4), April 2003. Last viewed on Jan 9, 2005 at http://www.dlib.org/dlib/april03/staples/04staples.html.

[41] M. Szeredi. File System in User Space README. http://www.stillhq.com/extracted/fuse/README, 2003.

[42] R. Tansley, M. Bass, M. Branschofsky, G. McClellan, and D. Stuve. DSpace system documentation. Documentation, MIT Libraries, August 10 2004.

[43] J. Teevan, C. Alvarado, M. S. Ackerman, and D. R. Karger. The perfect search engine is not enough: a study of orienteering behavior in directed search. In *Proceedings of the 2004 Conference on Human Factors in Computing Systems (CHI '04)*, pages 415–422. ACM Press, 2004.

[44] The Joint Information Systems Committee. Supporting digital preservation and asset management in institutions. http://www.jisc.ac.uk/index.cfm?name=programme_404, October 2004.

[45] The Long Now Foundation. Digital dark age: Digital data loss and preservation resources. http://www.longnow.org/10klibrary/darkage.htm.

[46] U.S. National Archives & Records Administration. National Archives names two companies to design an electronic archives. http://www.archives.gov/media_desk/press_releases/nr04-74.html, August 3 2004.

[47] w3c. W3C RDF Data Access Working Group. http://www.w3.org/2001/sw/DataAccess/, October 2004.

[48] w3c. W3C semantic web. http://www.w3.org/2001/sw/, October 2004.

[49] A.-I. A. Wang, G. H. Kuenning, P. Reiher, and G. J. Popek. Conquest: Better performance through a disk/persistent-RAM hybrid file system. In *Proceedings of the 2002 USENIX Annual Technical Conference*, Monterey, CA, June 2002.

[50] J. C. Worsley and J. D. Drake. *Practical PostgreSQL*. O'Reilly, 1st edition, 2002.