Title

A file by any other name: Managing file names with metadata

ISBN

9781450329200

Authors

Parker-Wood, A
Long, DDE
Miller, E
et al.

Publication Date

2014-06-30

DOI

10.1145/2611354.2611367

Peer reviewed

A File By Any Other Name: Managing File Names with Metadata

Aleatha Parker-Wood
Conservatoire National des Arts et Métiers
aleatha@cs.ucsc.edu

Darrell D. E. Long, Ethan Miller
University of California, Santa Cruz
darrell,elm@cs.ucsc.edu

Philippe Rigaux
Conservatoire National des Arts et Métiers
philippe.rigaux@cnam.fr

Andy Isaacson
NoiseBridge
adi@hexapodia.org

ABSTRACT

File names are one of the earliest computing abstractions, a string of characters to uniquely identify a file for the system, and to help users remember the contents when they look for it later. They are also a rich source of semantic metadata about files. However, this metadata is unstructured and opaque to the rest of the system. As a result, metadata in file names is often error-prone, and hard to search for. File names can and should be more meaningful and reliable, while simplifying application design and encouraging users and applications to provide more metadata for search.

We describe a POSIX compliant prototype file system, TrueNames, which demonstrates an alternate approach to naming and metadata, called *metadata aware naming*. TrueNames separates the task of uniquely identifying a file from the task of helping the user remember its contents. It captures metadata in a structured format for later indexing, and uses it to generate file names which are *correct*, *regenerable*, and *disambiguatable* by design. TrueNames simplifies application design by providing a consistent interface for metadata aware naming, incurs a low overhead of approximately 15% under realistic workloads, and can simplify a wide variety of data management tasks for both applications and users.

Categories and Subject Descriptors

E.5 [**Files**]: [Organization/structure]; H.3.2 [**Information Storage and Retrieval**]: Information Storage—*File organization*

General Terms

Design, Human Factors

1. INTRODUCTION

File names have existed since the earliest file systems, and serve two important functions. First, they serve to uniquely identify a file over time. Second, they serve to assist our memory, describing the contents of a file, and helping us to find it or recognize it when we look at it later. In order to help users to find and remember files, they often contain a bounty of useful metadata about the file.

However current approaches to file names have several flaws. Names are unstructured semantic metadata which is opaque to the system and unavailable for indexing. Formatting is up to the user, and is error prone and inconsistent, making it hard to find files later. The name can easily become out of sync with the file contents. Applications cannot effectively cooperate with each other or users to generate names. And changing a file name can destroy information that was previously available.

Consider this file name from the author's experiments:
`createfiles_HDD_truenames_100000files_1threads.data`
Looked at in one light, this is a long, arbitrary, error prone string of characters. In another light, it is a rich source of semantic metadata about the contents of the file that could be used for search and analysis, and without which the contents of the file will be useless. The challenges are to extract useful metadata, and make it simple to create and maintain user-friendly file names, all without creating additional work for users and applications.

To address these challenges, we propose to disassociate the two functions of names, separating the task of uniquely naming a file for applications from that of providing meaningful names for users. Files can be given user-friendly names automatically and in a structured fashion, based on metadata provided by users and applications. These file names have many advantages over conventional file names. They are *correct*, because they are continuously synchronized with the file's metadata. They are *regenerable*, allowing us to compress and recreate names at will without loss of information. They are *flexible*, allowing files to change names without breaking application references, and display different names in different contexts. File names can be *disambiguated* using all available metadata, which makes search results easier to interpret, and reduces accidental data over-writes. The metadata which we capture is structured, making it readily available for search and data management. We describe these names as *metadata aware* names.

Metadata aware file naming can make naming more reliable and less error prone. Many applications already offer some form of automatic file naming, and by making that functionality part of the file system, we can speed application development and reduce code duplication. Using metadata allows different applications to share responsibil-

SYSTOR '14, June 10 - June 12 2014, Haifa, Israel

ACM 978-1-4503-2920-0/14/06...$15.00
DOI 10.1145/2611354.2611367.

ity for managing file names, rather than having metadata and names locked into application silos. Being able to regenerate file names allows us to port files between file systems with differing constraints, generate a meaningful file name in each location, and then reconstruct the original file name whenever needed. Names can be customized based on the current directory, the user, or the search context. We can easily move metadata between file names and directory names, or store it for later use. And finally, by gently encouraging users and applications to share more metadata in a structured fashion, we can make more metadata available for indexing, to improve the quality of file system search or support a non-hierarchical file system.

As a proof of concept, we describe our prototype file system, TrueNames, a FUSE-based file system which provides a durable unique identifier for a file which can be used by applications, captures rich metadata in a structured format, and uses it to dynamically generate user-friendly file names using *templates*. TrueNames demonstrates the feasibility of metadata aware naming. It offers extensive new functionality, and incurs very low overheads, less than 15% on realistic workloads. Much of the additional cost is incurred by added kernel crossings, suggesting that an in-kernel implementation would have even lower performance impact, while significantly improving file system search and data management by increasing the structured metadata available for indexing.

2. USE CASES

Metadata aware naming can be used as a broadly applicable framework for solving cross-cutting concerns. It can be used by applications to simplify common tasks, and by scripts and end-users to better manage files. It can even help to prevent data loss caused by overwrites, and allow multiple applications to cooperatively name files. We describe a variety of use cases, and explain how metadata aware naming can benefit users and applications in each case.

2.1 Managing a photo collection

File names have lagged behind UIs in photography, making it challenging to find and manage photos outside an application. While most applications offer excellent GUI photo management, many use the default file name from the camera, usually a per-camera sequence number such as `DSC_12.jpg` or `IMG_655.jpg`. File name collisions are common, especially for owners of multiple cameras. The latest version of iPhoto [4] uses the default name, and names derivative files based on size, or an index corresponding to order of face discovery, such as such as `DSC_12_1024.jpg` or `DSC_12_face1.jpg`. iPhoto names cannot be managed by the user, and offer little information about their contents. Higher end applications such as Aperture [1] or Lightroom [5] allow the user to bulk rename files during import and export, using metadata such as Exif fields [10].

However, these applications cannot keep file names in sync if metadata changes, making it difficult to manage photos in more than one application. For instance, a user might want to use facial recognition from iPhoto, while touching up photos in Lightroom. If the user wants to find a retouched photo of a certain person, they cannot search for it using metadata, and if they wish to put a person's name in the file name, they must name the files manually, then reimport the new file name into one or both applications' databases.

TrueNames can significantly improve photo naming, by taking metadata extracted by the application and automatically constructing file names for photos based on metadata such as where they were taken, who was in them, and what size they are. If the metadata changes (such as a recognized face, or the addition of geo-tagging), TrueNames automatically updates the file name to reflect the correct information. File name collisions can be reduced, since photos from two cameras will have different metadata which can be used to generate useful names. In addition, TrueNames allows multiple applications that operate on the same files, such as iPhoto and Lightroom, to share responsibility for generating a meaningful name. Both applications can export metadata which can be used for naming and search, and TrueNames can manage the formatting and creation of file names, merging metadata from both applications into a single meaningful description of the file, something which is currently very difficult.

2.2 Managing experimental data

One common problem scientists face is managing experimental results. A scientist might run a series of experiments, varying some parameters and holding others constant, outputting the results to a file, then decide based on the results to vary other parameters. This is commonly managed by creating file names programmatically. However, this approach has drawbacks. For instance, if the user wants to search the results by parameter, they will need to use string matching or a regular expression. These both rely on consistent formatting, such as using the same field separator and the same field order. A metadata field which contains the same character as the chosen field separator (such as an underscore in a library name) can throw a regular expression off. Finally, common queries such as range searches require complex regular expressions to perform.

Another common problem occurs when rerunning experiments while setting additional parameters. The user must remember to change the file name to reflect the new parameters, and older file names will not have the new parameters, even if they were applicable. If the user forgets to change the code which generates file names, they may overwrite existing results, wasting hours or days of compute time.

TrueNames simplifies creating file names and searching for metadata. Rather than programmatically generating file names, experiments can simply add metadata for all the parameters. File names can be generated using a simple template, and then the user can expand the template as new parameters become relevant, resulting in both old and new files using the new template. Collision detection can prevent overwriting by disambiguating files on the fly. The metadata fields are structured and easy to index and search, regardless of what characters occur in the field, or the order fields occur in the file name. File names are an accurate reflection of their contents. The authors used TrueNames to manage experimental results for this work, and found it to be extremely helpful and simple to use, allowing them to easily manage multiple experiments and parameters, generate accurate and meaningful file names without code changes, and then search their results later.

2.3 Recreating file names

File name portability is a known problem in computing. For instance, some file systems support longer names and

paths than others. When moving files from one system to another, file names can be truncated, losing important information, and making it difficult to find a file, even if it is brought back later. TrueNames can help with this, by allowing a file name to be regenerated based on the metadata. For instance, files being moved to more restrictive file systems can use a different name template with fewer fields, creating an appropriately sized name without truncating the file name at an arbitrary point, and preserving the information the user finds most important. If the file is retrieved later, the original file name can be recreated exactly. Since structured metadata was stored and used to generate the original name, it can also be used to help create meaningful directory names, in essence pivoting metadata between file names and directory names.

2.4 Search and semantic file systems

Metadata aware naming provides a convenient way to organize a file hierarchy based on the context, and to name files according to the facets that need to be emphasized in a particular context. This is of particular importance in semantic and non-hierarchical file systems, such as SFS [15], LISFS [19], Cador [11] and Inversion [18]. Rather than having a single path to a file, these systems offer multiple virtual hierarchies based on metadata, where each directory is a query result. They can be backed by conventional file systems [15], or a file system-like interface can be backed by a database which contains BLOBs and associated data [11, 18, 19].

During search, metadata aware naming can give names to each file based on its context and the query. For instance, in a photo marketplace based on a BLOB storage backend, the same photo can be named on the owner's personal computer using places, faces and so on, whereas it should be shown in the shared marketplace area with metadata of interest to potential buyers (e.g., provider, price, and subject).

Even in hierarchical file systems, search often strips directory context, giving you a list of files which may all have the same name. This becomes even more problematic in a semantic file system, where there are no directories for context. For instance, if we name photos using who, when, and where, and then query for "photos of Mary in Hawaii on July 7, 2011", then we may have ten files named `Mary_7-7-2011_Hawaii.jpg`. This is an area where disambiguation can shine. If some or all the fields in the template are already fixed by the query, more metadata can be added to the file name, allowing users to distinguish between results at query time.

3. ARCHITECTURE

Having file names which can change outside the control of users and user applications poses a number of unique and interesting challenges. One must choose a storage mechanism for the metadata used for naming. There must be a way to define the structure of names, and what metadata they will use. Applications require a durable way to reference files, in order to maintain internal databases and repeatedly reference files. There must be a way to prevent accidental data loss through file name collisions. We discuss these challenges, and the modifications our prototype makes to support metadata-aware naming throughout the file system and applications. The architecture of our prototype is shown in Figure 1.

3.1 Storing metadata

The goal of our prototype was to store rich metadata for file names in a way that was portable, did not significantly change the semantics of the file system, and was easy to understand and manage. To that end, we selected the extended attribute interface for managing metadata. Extended attributes provide a simple key-value interface, and are associated with the inode (either by a reference to a metadata block or resource fork, or in the case of small metadata on ext4, directly stored in the inode), which means that files with hard links, which contain the same data, can also share metadata and names. (Soft links continue to offer the ability to have a mix of automatic and manual names for a single file.) Many applications already use extended attributes, they require no additional libraries, and are supported by most modern file systems, improving portability of our ideas. Many other solutions are possible, and we discuss some of them in the future work section.

3.2 Managing names

TrueNames assumes that file names have *schemas*, a set of rich metadata which is broadly applicable to many different files. However, not all files of the same extension are assumed to have the same schema. For instance, an PDF file may be a graph of experimental results, or an e-book. A text file might be a configuration file, or a letter to a relative. These will have different schemas that are appropriate. Likewise, two files with differing extensions, such as `.jpg` and `.gif` files, may have a schema that is appropriate for both. Thus, we allow a file to reference a *template*, which is a string containing each of the metadata fields of the schema. A template has a name, and defines the structure of a file name that is appropriate for that file, as shown in Example 1. Templates can contain file extensions which serve as the default extension. However, if the user supplies a file extension, it will override the template extension, allowing different types of files to share templates. For instance, `.jpg` and `.gif` files can share a `photo` template.

Example 1 Template file

```
music {$artist}-{$album}-{$song}.mp3
photo {$seq}_{$date}_{$camera}_{$location}.jpg
paper {$author}_{$conference}${year}_${tags}.pdf
exp {$wkload}_{$files}files_{$threads}threads.data
```

Templates are associated with a file using an extended metadata field called `user.naming.type`, which references the name of a template. This field is not mandatory, making it possible to mix manually and automatically named files throughout the file system. If a user or application wishes to rename a file, they can change `user.naming.type` to the name of a different template, adding additional metadata as needed. If a user wishes to manually manage a name, they can remove the current template from the file's metadata, or not choose a template during file creation.

As metadata is added or updated by users and applications, the file name is updated to reflect the current state of the file's metadata. This entails storing the extended metadata, looking up the template, re-calculating the file name, and then renaming the file. It may also entail handling file name collisions caused by the rename.

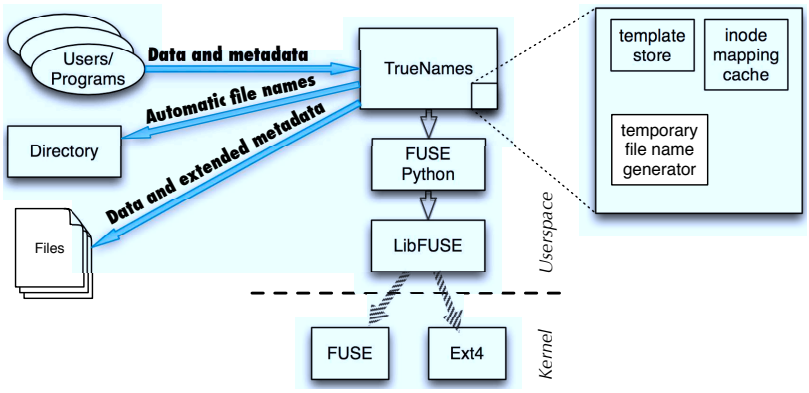


Figure 1: Architecture of TrueNames

If not all of the metadata referenced is available, TrueNames makes a best effort to update the file name, populating all of the known fields, and marking metadata that is missing with a default value. In our photo example, that might result in a file name of `DSC1967_7-21-13_NikonD50_{$location}??.jpg` for a photo that has not yet been geotagged.

In our prototype, templates are stored on a global basis, and loaded on file system startup. However, a file system might wish to allow individual users or directories to have associated templates, and for files to have different names for different users (another possibility created by disassociating system names from user-friendly names). We plan to explore personalized naming in future work.

3.3 Programming with changing file names

One complication created by automatically named files is that the file name can change between accesses or modifications. For instance, when metadata is being added to the file, each new metadata item which is used by the template will trigger a rename. In these cases, the user or application needs a durable way to reference the file they are updating. The key to metadata aware naming is the ability to disassociate the name the user sees from a unique identifier which serves as a durable reference for programs and the system. In file systems which are willing to deviate from the POSIX specification, file creation could return a durable reference such as a GUID or object ID directly. However, our prototype demonstrates a fully POSIX compliant solution which requires minimal application changes.

As a proof of concept, we use the inode to guarantee uniqueness of references. In the next version of TrueNames, we will add the inode's generation number, similarly to how NFS handles stale file handles [21], to guarantee that the inode has not been freed and reused. We allow files to be referenced either by file name or directly by inode number, as if the inode number were a file name.

To reference a file in a directory by its inode number rather than by name, it can be opened using a reference to `<dirname>/.inode/<inodenumber>`, as shown in Example 2. This allows programs and scripts to create an automatically named file, call `fstat` on the file handle to acquire a durable reference, and then add metadata, all without needing to know the file's name.

Example 2 Reference by inode

```
$ curl http://indyband.org/1.mp3 > template=music
$ ls -i
1317 {$artist}??-{$album}??-{$song}??nozwZ8.mp3
$ setfattr -n user.artist -v"Indy Band" .inode/1317
$ setfattr -n user.song -v"So Obscure" .inode/1317
$ setfattr -n user.tracknum -v"1" .inode/1317
$ ls
Indy Band-{$album}??-So Obscure.mp3
```

Under a typical file system such as ext4, it is not fast for an application to access a file by its inode number. Accessing a file by its inode number requires searching the system for a file which has a matching inode number and then resolving it to a name, which can be very expensive. While we already reduce the cost by reducing the search space to a single directory, this still requires an linear scan over the directory inode. To get acceptable performance, we further reduced this cost by adding an inode cache map to TrueNames, which allows us to look up a file name in constant time given an inode. This is similar to the name cache used by the Linux VFS, although the lookups occur from inode to name, rather than vice versa. The inode map is populated from the directory inode on first access, and then caches all inodes in that directory up to some threshold (ten thousand files in our experiments.) We use a mark and sweep policy, and evict up to half the entries from the cache when full.

By encouraging applications to use a static reference, users can modify user-friendly file names at will, and files can have context dependent names, without breaking existing application references. Static references work for all files, not just automatically named ones, so even manually named files can benefit from this feature.

3.4 File creation with automatic names

In order to create a file in most file systems, we need two pieces of information. First, we must signal what directory we are creating a file in. Second, we must make sure that the file name we are creating is unique, to prevent overwriting an existing file. Automatic naming adds two additional problems. We need to signal what template we would like to use. And since no known file system allows you to add metadata before creating a named file, we cannot use metadata to generate the initial file name.

Our goal for the prototype was to maintain existing semantics as much as possible, and require minimal rewrite of existing code, which ruled out adding or modifying system calls. We therefore overrode the semantics of the existing system calls, `open()`, `creat()`, and `mknod()`, such that if a file is created with a name which ends in `template=X`, where `X` matches the name of a known template, the name is managed by the file system, and a new file is generated with an automatically generated unique file name. The initial file name is composed of the template contents, followed by a unique alphanumeric suffix, and finally, any file extension, as shown in Example 2. We use atomic file creation with `O_EXCL` internally, to prevent race conditions, as well random back-off retries if we receive `EEXIST` during initial file creation. Once the file is created, metadata can be added to populate the file name. Since metadata must be added one field at a time, there is a possibility of a temporary name collision during metadata addition, which we attempt to detect and handle.

Alternatively, the application can create a file name using any name it wishes (including one given by the user), and then set the template and metadata fields after file creation. The original file name can also be stored in metadata, and then reapplied later, or used for resolving collisions.

3.5 Handling collisions

One problem that can arise with automatic naming is that two files in the same directory may be different in content and metadata, but share all the fields that are currently referenced in the template. During rename, we check to see if the new metadata results in two files with the same name. If so, we check to see if any metadata differs. If disambiguating metadata is available, we add the first available metadata to the file name which will disambiguate the files, along with its extended metadata key. If not, then and only then, do we overwrite the existing file. In the future, we plan to explore more user-friendly strategies for disambiguation.

3.6 Application level support

A file system which requires extensive changes to applications is unlikely to see adoption. We built TrueNames to show how a standard POSIX compliant file system could expose new naming functionality, but newer file system types such as object stores and non-hierarchical file systems can easily add new APIs for bulk metadata management and nameless file creation, which would ease implementation of metadata aware naming.

Existing applications which don't wish to take advantage of the added functionality can run on TrueNames without any modifications, and see very little change in performance. In order to take advantage of the new functionality, applications need to create or reference a template, and begin exporting metadata for each new file. Optionally, they can begin referencing files using an inode reference. We describe the changes to semantics in our prototype, and how they affect applications.

open/creat/mknod As noted, if these calls are invoked using a directory path followed by `template=X`, they create an automatically named file using the template, a unique suffix, and an extension, and set `user.naming.type`. Otherwise, they will create a file in the normal fashion.

setxattr/removexattr In addition to setting and removing extended attributes, these calls now trigger a recalculation of the file name. If the attribute set or removed is one present in the file template, then the file will be renamed. These can also be used to change the template or freeze the current name, by setting or removing `user.naming.type`.

rename This operation can be used in a variety of ways.

- If the target path contains a different directory, but the same file name, the file is moved to the new directory using its current file name.
- If the target path contains a different file name which is the name of an existing template, we update the file to use the new template.
- If the target path contains a different file name which is not the name of an existing template, we assume the user wishes to manually control the file name. We apply the new file name, and remove `user.naming.type` from the file's metadata.
- In all cases, if a new inode is created during rename (for instance, if the file is migrated between file systems to a file system which supports extended attributes), all the extended metadata is copied to the new inode.

Hard links Under our prototype, a hard link shares an inode, and therefore all extended metadata, with the path it links to. Therefore, files which are hard links to an automatically named file will share a name with its target. A hard link to a dynamically named file in in the same directory is not feasible, since it has the same name as the target, and will fail with `EEXIST`. Otherwise, hard links function as expected. This is a limitation of our prototype. In future work, we plan to investigate other methods for automatically managing both hard and soft links.

Soft links Soft links work as before, and can target either a file name, or an inode path, depending on the desired behavior. Soft links can be used to supply multiple names in the same directory, such as an automatic name and a manual name. Due to restrictions on extended attributes, soft links cannot be automatically named in our prototype.

Every file in our prototype now has at least two names: its human readable name, either auto-generated or assigned by a human, and its inode number preceded by its directory path. It may have additional names via hard and soft links. Human readable names and inode references are interchangeable in all file system calls. A reference by inode can be opened, linked, have metadata set or gotten, and so on. However, `.inode/` itself is not a real directory on disk, and cannot be opened or iterated over.

If multiple applications manage the same files, then there is the possibility of both applications attempting to manage the template and metadata. Adding additional metadata to file names and templates can only benefit naming and search, but applications may wish to prompt the user before changing the template or removing fields.

4. EXPERIMENTAL DESIGN

We have demonstrated how new functionality can be added to the file system to make it more searchable, to make file names more correct and structured, and how this functionality can easily integrate into existing file systems. However, a file system's functionality must also be balanced against its performance. We describe how we evaluated the performance of TrueNames. In order to effectively benchmark such a file system, we need to answer questions on how it affects basic file system operations, such as file creation, deletion, and renaming. We also need to describe the effect on extended metadata operations. In order to evaluate our file system, we compared it against two other file systems, one comparable Python FUSE file system, as well as a raw ext4 file system in order to characterize the overhead of FUSE and Python versus the overhead of our file system.

- `xmp` is the example file system which ships with fusepython. We added support for extended attributes, using the same library, `py-xattr`, used by TrueNames. Otherwise, it is a vanilla FUSE file system.
- As a point of reference, we also include file system performance on a raw ext4 file system.

We ran both TrueNames and `xmp` in single threaded mode, since `xmp` does not support multiple threads by default, and we wanted to modify it as little as possible. TrueNames will support multiple threads in the future. We disabled the inode cache, and set the `entry_timeout` to 0 in order to prevent stale file name entries.

We ran two batches of experiments, one using an SSD, and one using a hard disk drive. Our SSD experiments were run on a 100 GB Intel 330 Series SSD, in an 8 core Intel Xeon CPU E3-1230 V2 @ 3.30GHz with 16 GB of RAM. Our HDD experiments were run on a 7200 RPM 500 GB Seagate Constellation drive, in an 8 core Intel Xeon CPU E5620 @ 2.40GHz and 24 GB of RAM. In both cases, we ran on Fedora with a 3.9.10-200.fc18.x86_64 kernel, and an ext4 file system as our backing store.

5. RESULTS AND ANALYSIS

TrueNames is a proof of concept user space prototype, but even so, it has very low overhead, demonstrating that automatic naming can be added to production file systems with minimal performance implications. We tested TrueNames under a variety of micro and macro benchmarks, in order to analyze its performance under extreme conditions as well as realistic workloads. TrueNames is noticeably slower under our micro benchmarks, as expected, but the performance penalty can be measured in fractions of a millisecond, and TrueNames exhibits no scaling issues under high load. Under normal file system loads, such as our macro benchmarks, TrueNames performs with only a 15% overhead. In addition, TrueNames shows no impact on operations other than file creation and extended metadata operations, which are rare in most workloads. In aggregate, these numbers show that metadata aware naming is suitable for many file systems, large and small alike, adding useful new functionality at little performance cost.

5.1 Microbenchmarks

There are a number of benchmarks designed to exercise metadata. However, these are generally aimed towards file system metadata, such as performing high-speed updates to modification times. Tools such as Filebench [2], while quite flexible, do not offer the ability to modify extended attributes out of the box. By contrast, we needed to evaluate our system's impact on extended metadata performance. In order to do this, we wrote a benchmark designed to add, update, and delete extended attributes continuously. In effect, if the file is of an automatically named type, this results in TrueNames continuously renaming the file.

In order to focus as much as possible on the metadata speed, we pre-created a file set of size n. We then iterated over the file set until all n files had been touched, setting a single extended attribute on each file, calculating the latency of each operation and collecting statistics. Ext4 stores small metadata attributes in the inode, so files with a very large number of extended attributes may show lower performance than our benchmarks. One potential optimization is to ensure that metadata attributes used in the name are kept in the inode, since they are likely to be small, and not numerous.

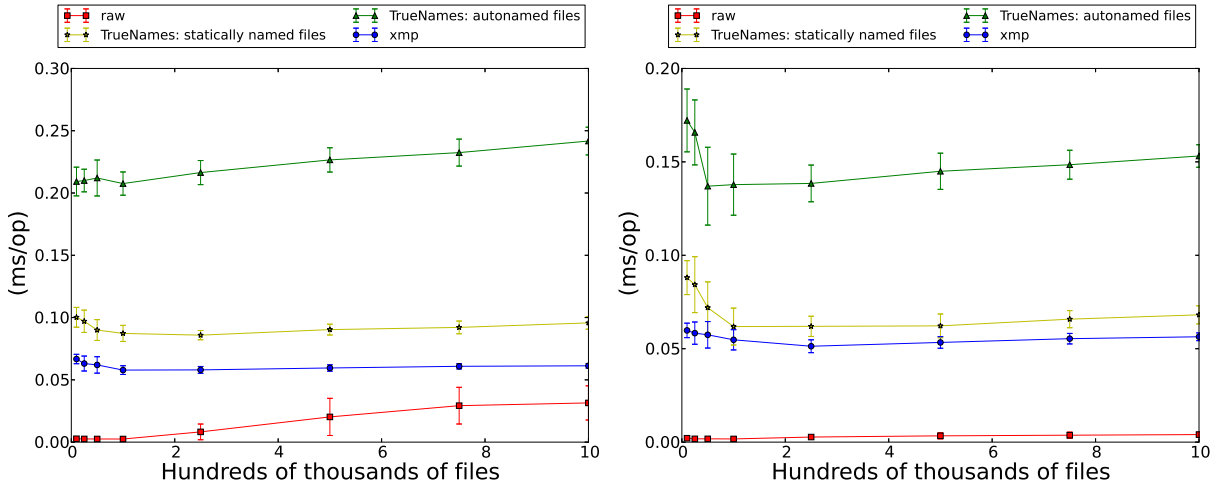
Once the add benchmark completed, we ran similar benchmarks to update an attribute, and finally, to delete an attribute. This allowed us to exercise the new code paths continuously, highlighting any performance differences from our baseline file systems.

We ran this benchmark for both automatically and manually named files, in order to quantify the overhead incurred both with and without using the new features. We ran the metadata add, update and delete benchmarks for file sets from 10,000 to 1,000,000 files, which is comparable to modifying every file on a modern laptop at once. We then repeated each test forty times in order to smooth noise and calculate a standard deviation.

Microbenchmarks are the most intensive tests, so it is unsurprising that they show the largest difference between the file systems. Examining the difference between automatically named files and manually named files, we can see that automatic naming incurs a 100% penalty over metadata operations which do not affect the name, across add (Figure 2), update (Figure 3), and delete (Figure 4), and for both HDDs and SSDs. However, even at a 100% penalty, the additional cost can be measured in fractions of a millisecond. Looking at the difference between the baseline fuse system `xmp`, and TrueNames without name templates, we can see that there is approximately a 30% overhead to using TrueNames without automatic naming. This is primarily due to checking every time if the file has a template, which requires retrieving extended metadata, and therefore both an additional kernel crossing and potentially a disk access. In total, TrueNames adds four extra kernel crossings per one file creation. The additional kernel crossings are a performance issue specific to FUSE, and would not occur in an in-kernel file system.

Even at this high operation rate, and for a million files, we can see from the latencies that the disk cache is rarely saturated. This performance will occur in a small fraction of operations, usually during file creation, and the additional latency will be masked under most normal workloads, as we discuss in the next section. TrueNames shows a fixed overhead without scaling bottlenecks up to at least a million files, making it suitable for fairly large workloads.

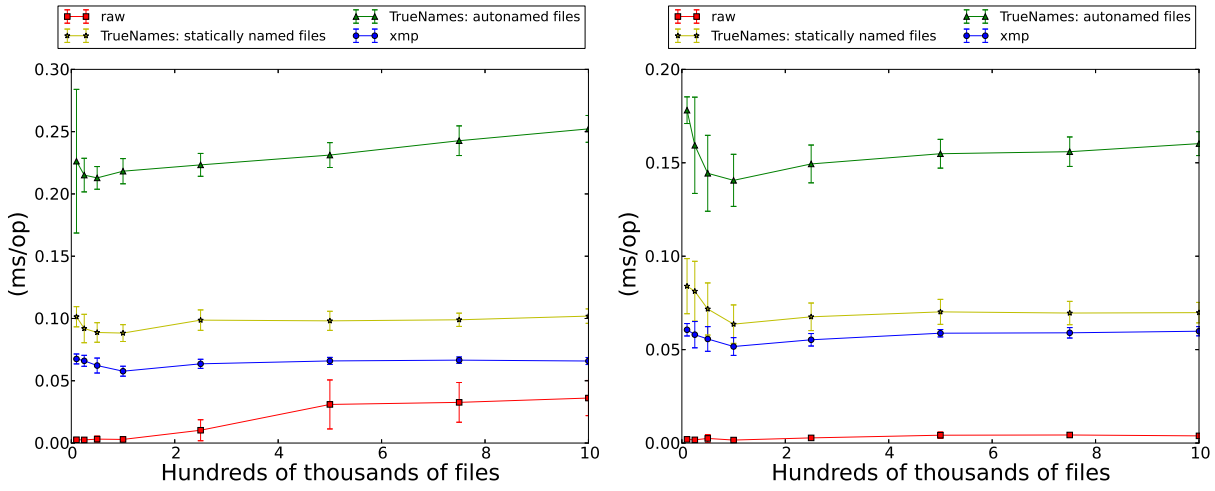
5.2 Macrobenchmarks



(a) HDD: Adding metadata

(b) SSD: Adding metadata

Figure 2: Latency and standard deviation per add extended metadata operation, for 40 runs of the add benchmark.



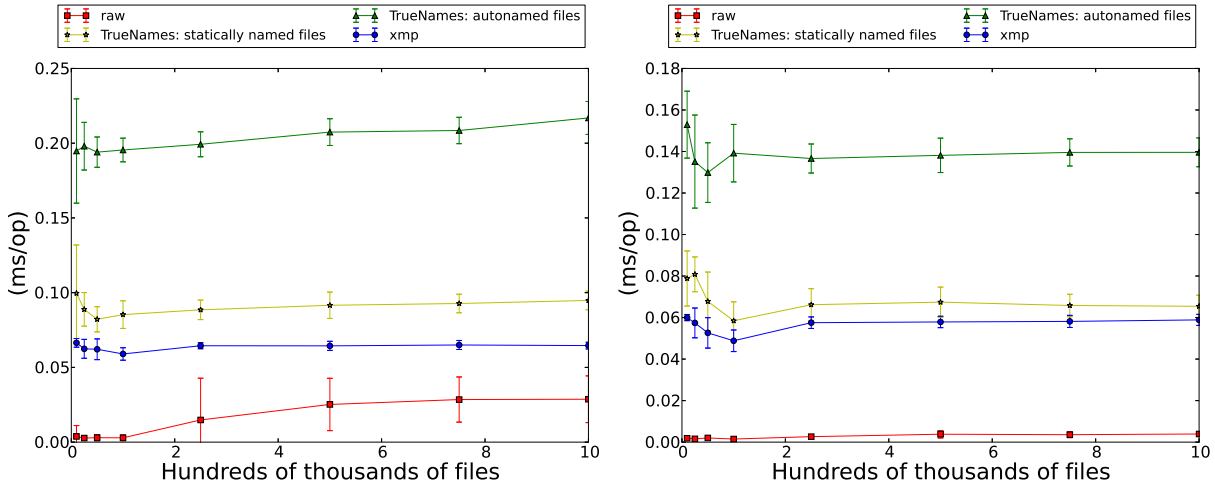
(a) HDD: Updating metadata

(b) SSD: Updating metadata

Figure 3: Latency and standard deviation per update extended metadata operation, for 40 runs of the update benchmark.

While micro benchmarks can be useful for setting an upper bound on performance, they are often an unrealistic assessment of how a file system will perform in practice. In the real world, file systems are experiencing a variety of operations from many different sources. In order to simulate the performance in a realistic environment, we chose a standard benchmark, Filebench [2]. This benchmark does not exercise the extended metadata functionality, and is therefore comparable to the statically named files experiments from section 5.1. We ran two different benchmark suites, the `fileserver` suite, which is designed to simulate the behavior of a typical file server, by performing a series of creates, deletes, appends, reads, writes and attribute operations on a directory tree. Mean directory size is 20 files, and the mean file size is 128kB. The workload generated is somewhat similar to SPECsfs [3]. We also ran the `createfiles` benchmark, which creates a specified number of files in a directory tree, with an average directory size of 100 files. File sizes are chosen according to a gamma distribution with a mean size of 16kB. We varied each of these from 1 to 32 threads, and from 100,000 to 1,000,000 files. We then repeated each test forty times, in order to smooth noise and generate a standard deviation.

For both the fileserver and createfiles benchmarks, TrueNames has file creation performance that is highly comparable with that of `xmp`, at about 15% overhead. File creation performance for the fileserver benchmark can be seen in Figure 5. File creation performance for createfiles is in Figure 6. Other typical operations, such as writing a file, deleting a file, or calling `stat` on a file, had insignificant overhead, meaning that TrueNames is suitable for all but the most create-intensive workloads. Most of the fixed overhead of TrueNames is masked by normal operation latencies.



(a) HDD: Deleting metadata

(b) SSD: Deleting metadata

Figure 4: Latency and standard deviation per delete extended metadata operation, for 40 runs of the delete benchmark.

6. RELATED WORK

Automatically naming files is an under-explored area. The most similar areas of research are those of application-generated names, web search snippets, and non-hierarchical file systems, each of which we discuss.

6.1 Application-generated names

As discussed in Section 2, many applications such as iTunes [8], iPhoto [4], and Mendeley [6] currently generate file names, either for their own use or that of the user. However, none of them offers a generalized framework for file naming, and do not reflect outside changes to metadata. By contrast, TrueNames offers automatic naming as a service which any application can use, simplifying application development, keeping names synchronized with metadata regardless of source, and encouraging developers to export structured metadata.

6.2 Naming on the web

File naming is analogous to disambiguating search results on the web. Web search, like file system search, has many files with unique resource identifiers, and when returning results the search engine must help the user choose between them. Much of the web is text, and the common approach is to reveal a snippet from the page containing the search terms. Newer search types, such as video, rely on a title and key frame. However, in file systems, these approaches are not feasible. Many files are in opaque data formats, and no text snippet or key frame is available. Thus, we rely on file type-specific metadata to help the user identify their files.

6.3 Non-hierarchical and semantic file systems

The problem of automatically naming directories based on metadata has been heavily explored. In particular, non-hierarchical file systems [7, 15, 16, 18, 19, 25] present directory names based on the metadata of files, allowing the user to navigate and select files using their metadata. For instance, the original Semantic File System (SFS) [15] treats all directory names as queries, and automatically generating subdirectories based on lists of attributes and values. The Linking File System [7] generates subdirectories based on links. Similarly, The Logic File System (LISFS) [19] allows some attributes to subsume others. If the results to a query contain one or more subsumed attributes, only the higher level attribute will be displayed as a directory name, and only attributes which distinguish between the query results are shown. This is similar to our disambiguation method. However, we rely on the template for a name, and add metadata only when required by a name collision.

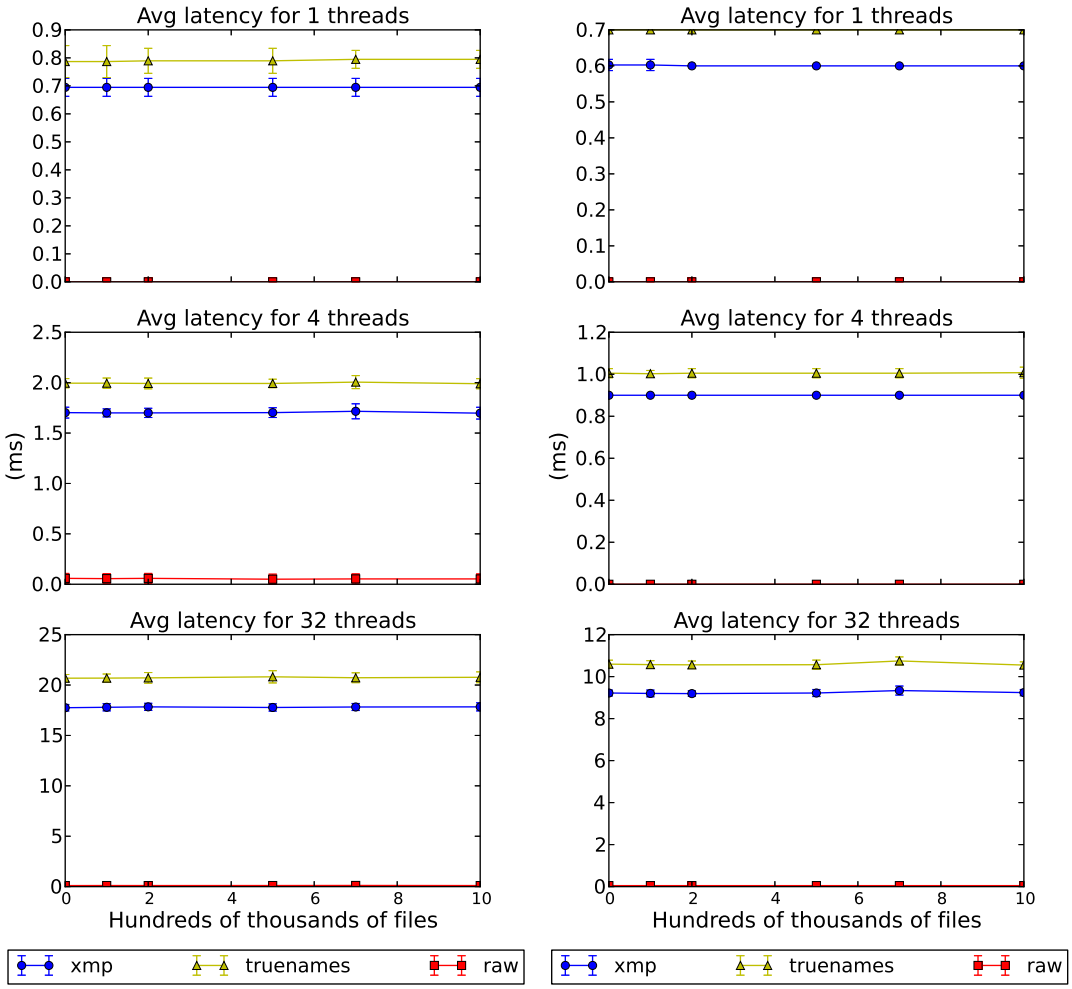
None of these focus on file names, instead focusing on naming directories as a way to create queries over documents. Our approach is designed to complement non-hierarchical systems, providing a generic mechanism for files to be easily recognized regardless of context, and allowing non-hierarchical systems to disambiguate file names at query time.

The most similar work to TrueNames is that of Jones et al. [17], who proposed a non-hierarchical HPC file system with automatically generated file names, chosen by examining the distribution of metadata fields. By contrast, our work uses a more robust and less complex scheme which puts the user and application in control of which metadata is used, and allows them to select attributes which are most appropriate for the file's semantic type, rather than relying on statistical techniques.

6.4 Other file systems

Our work on collision detection also has implications for systems with very large directories, making it similar to systems such as Giga+ [20]. One of the challenges for scalable directories is being able to create a large number of unique file names quickly, and in future work, we will examine ways to make TrueNames scale to large distributed directories.

Object stores [9, 13, 14, 12, 24] allow a file to be addressed directly by an object identifier, rather than as a series of logical blocks, much as TrueNames allows files to be referenced directly. Services such as OSMS [22] for OpenStack Swift provide metadata storage and search. Since object IDs are fixed identifiers for data objects, stores such as these can use metadata aware naming for user friendly names even more easily than block based storage, and offer better facilities for bulk metadata management.



(a) HDD

(b) SSD

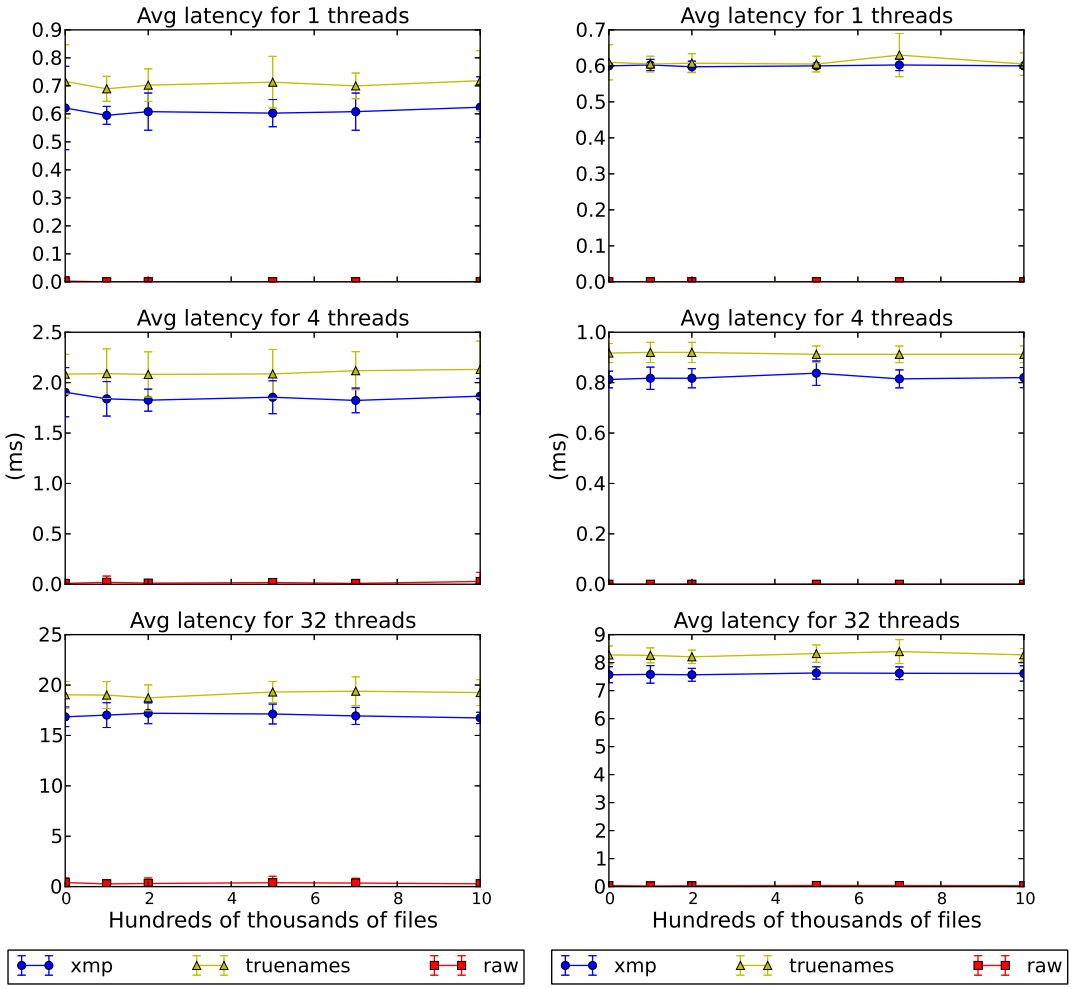
Figure 5: File creation times and standard deviation for 40 runs of the createfiles benchmark on HDD and SSD

7. FUTURE WORK

TrueNames was designed with non-hierarchical file systems in mind. In the future, we intend to extend TrueNames to a non-hierarchical context, where metadata is used to find files and display them. In a search context, TrueNames can help disambiguate search results, provide additional metadata, and potentially expand or collapse file names dynamically, based on a list of useful fields provided by the user. We are particularly interested in the potentials of object storage. We note that some minor modifications to the POSIX specification, allowing a user or application to simultaneously create a file and add metadata, and get a file name and file handle back as the return value, would be significantly reduce the possibility of collisions and improve performance. Bulk metadata addition would also improve performance and stability.

We are interested in doing user studies with TrueNames to confirm its usefulness in real world situations. In the future, we plan to make it more robust, adding support for multi-threading and other performance and stability enhancements, as well as porting it to additional platforms. We will add support for templates on a per-user basis. This allows users and applications to define more personalized templates, as well as providing a higher degree of privacy around metadata fields. Templates will be stored and accessed based on the `uid` of the calling application, and user templates will override global templates of the same name. We will also investigate ways of providing more flexibility for naming hard-linked files, such as associating templates with directories, and setting a default template for an entire directory, such as an images folder.

Finally, we intend to explore the implications of using TrueNames in distributed file systems such as Lustre [9] and Ceph [24], where file names and extended metadata are stored on multiple servers, adding new challenges around scalability and metadata management. Large scale systems for science can benefit greatly from better metadata and file name management.



(a) HDD

(b) SSD

Figure 6: File creation times and standard deviation for 40 runs of the fileserver benchmark on HDD and SSD

8. CONCLUSIONS

We have created a new method of file naming, *metadata aware file naming*. Our proof of concept prototype, TrueNames, adds a low 15% overhead under a variety of workloads, and scales up to a million or more files, demonstrating the feasibility of this approach. Metadata aware file naming provides a easy to use and efficient set of abstractions for managing file names and metadata. It eases file management by automating the process of file naming, and giving files context-aware and automatically updated file names. It also gently encourages users and applications to supply more structured metadata to improve the quality of search. Using metadata aware file naming can not only benefit users and application developers in the short term, it can ease the migration path to non-hierarchical file systems. By separating unique system identifiers from human readable file names, we enable file systems and users to work together to manage files and names more effectively.

Metadata aware file naming is broadly applicable, as we have shown by describing a variety of existing use cases. It can simplify application development, reduce data loss, and make it easier for users and applications to find and manage files. We have found our prototype, TrueNames, to be an extremely useful tool during the writing of this work, and will be porting it to OS X in the near future in order to take advantage of it on more of our machines.

Acknowledgements

This research was supported in part by the FUI Polymathic Project 2011-2014, the National Aeronautics and Space Administration, the Department of Energy, the National Science Foundation under award CCF-0937938 and IIP-1266400, and by industrial members of the Center for Research in Storage Systems. We also thank the industrial sponsors of the SSRC.

9. REFERENCES

[1] Aperture. http://www.apple.com/aperture/what-is.html.

[2] Filebench. http://sourceforge.net/projects/filebench/.

[3] Filebench wiki: Pre-defined personalities. http://sourceforge.net/apps/mediawiki/filebench/index.php?title=Pre-defined_personalities.

[4] iPhoto. http://www.apple.com/ilife/iphoto/.

[5] Lightroom Help: The Filename Template Editor and Text Template Editor. http://helpx.adobe.com/lightroom/help/filename-template-editor-text-template.html.

[6] Mendeley Add & Organize. http://www.mendeley.com/features/add-and-organize/.

[7] Ames, S., Bobb, N., Greenan, K. M., Hofmann, O. S., Storer, M. W., Maltzahn, C., Miller, E. L., and Brandt, S. A. LiFS: An attribute-rich file system for storage class memories. In *Proceedings of the 23rd IEEE / 14th NASA Goddard Conference on Mass Storage Systems and Technologies* (College Park, MD, May 2006), IEEE.

[8] Apple Inc. iTunes. http://www.apple.com/itunes/overview/, Jan 2010.

[9] Braam, P. J. The Lustre storage architecture. http://www.lustre.org/documentation.html, Cluster File Systems, Inc., Aug. 2004.

[10] Camera & Imaging Products Association (CIPA), and Japan Electronics and Information Technology Industries Association (JEITA). Exchangeable image file format for digital still cameras: Exif Version 2.3. http://www.cipa.jp/std/documents/e/DC-008-2012_E.pdf, Dec. 2012.

[11] Constantin, C., du Mouza, C., Rigaux, P., Thion, V., and Travers, N. A Desktop Interface over Distributed Document Repositories. In *International Conference on Extending Database Technology (EDBT'12)* (March 2012), pp. 104–107. Demonstration.

[12] Foundation, O. Swift architectural overview. http://docs.openstack.org/developer/swift/overview_architecture.html, Feb 2014.

[13] Ghemawat, S., Gobioff, H., and Leung, S.-T. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)* (Bolton Landing, NY, Oct. 2003), ACM.

[14] Gibson, G. A., Nagle, D. F., Amiri, K., Chang, F. W., Feinberg, E. M., Gobioff, H., Lee, C., Ozceri, B., Riedel, E., Rochberg, D., and Zelenka, J. File server scaling with network-attached secure disks. In *Proceedings of the 1997 SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Seattle, June 1997), ACM.

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

[16] Gopal, B., and Manber, U. Integrating content-based access mechanisms with hierarchical file systems. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI)* (Feb. 1999), pp. 265–278.

[17] Jones, S., Strong, C., Parker-Wood, A., Holloway, A., and Long, D. D. E. Easing the Burdens of HPC File Management. In *Proceedings of the 6th Parallel Data Storage Workshop (PDSW '11)* (Nov. 2011).

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

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

[20] Patil, S., and Gibson, G. Scale and concurrency of GIGA+: File system directories with millions of files. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST)* (Feb. 2011).

[21] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and Noveck, D. NFS version 4 protocol. IETF Network Working Group RFC 3010, Dec. 2000.

[22] Thomas, L. HTTP REST API for OpenStack Object Storage Metadata Search (OSMS). https://wiki.openstack.org/w/images/8/82/OSMS_API_v0.8_external.pdf, November 2013.

[23] Vinge, V. *True Names*, vol. 5 of *Binary Star*. Dell, 1981.

[24] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E., and Maltzahn, C. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)* (Nov. 2006).

[25] Xu, Z., Karlsson, M., Tang, C., and Karamanolis, C. Towards a semantic-aware file store. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS-IX)* (May 2003).