Dynamic Non-Hierarchical File Systems for Exascale Storage

Darrell Long (PI) and Ethan Miller (Co-PI)
University of California, Santa Cruz

Overview

Modern high-end computing (HEC) systems must manage petabytes of data stored in billions of files, yet current techniques for naming and managing files were developed 40 years ago for collections of thousands of files. HEC users are therefore forced to adapt their usage to fit an outdated file system model and interface, unsuitable for exascale systems. Attempts to enrich the interface, such as augmentation or replacement with databases, or the layering of additional interfaces and semantic extensions atop existing filesystems result in performance-limited systems that do not adequately scale.

Parallels exist between HEC systems and the web, where locating and browsing data sets has rapidly become dominated by search. The strengths and weaknesses of the web provide several useful lessons from which we have learned: 1) Although the web implements a hierarchical namespace, search has become the dominant navigation tool in the face of the massive volume of data that is accessible; 2) While finding *some* information is easy, finding the *right* or *good* information is not; 3) The easier it is for people to contribute information to a repository, the more critical it becomes to determine the veracity of that data; 4) The links that relate documents provide valuable insight into the importance of documents. From these observations we can see that simply modifying existing high performance filesystems to support search, and the requisite storage of additional semantic metadata, would be woefully inadequate.

We propose to develop a radically different filesystem structure that addresses these problems directly, and which will leverage provenance (a record of the data and processes that contributed to its creation), file content, and rich semantic metadata to provide a scalable and searchable file namespace. Such a namespace would allow the tracking of data as it moves through the scientific workflow. This allows scientists to better find and utilize the data *they* need, using both content and data history to identify and manage stored information. We take advantage of the familiar search-based metaphor to provide an initial easy-to-use interface that enables users to find the files they need and evaluate the authenticity and quality of those files. Realizing this vision requires research success in dynamic, nonhierarchical file systems design and implementation, large-scale metadata management, efficient scalable indexing, and automatic provenance capture.

Problem: Hierarchical Namespaces are Unnavigable for HEC Systems

Solution: Dynamic Nonhierarchical File Systems

A critical issue for managing files in HEC file systems is ensuring that users can easily find the files they need. We are proposing a search-based interface as the *only* interface to the file system name space. Our new architecture improves semantic file system performance and scalability by providing a combined indexing and storage layer on which semantic name spaces can be built. Thus, we must ensure that the interface is easy-to-use and has sufficient expressiveness to allow the construction of queries to pose complex but useful questions.

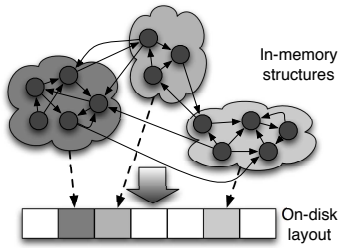

Figure 1: **Metadata clustering example.** Metadata clusters, partitions, shown in different shades, index different partitions of file system metadata. Each partition is stored sequentially on disk. The entire file system metadata index is composed of the set of all metadata clusters.

Web users are familiar with the problem of "information overload" in response to a search query; we will reduce this problem in our system by facilitating searches that are restricted to a local region of the provenance and relationship graph. This combination of file relationship information and per-file metadata has strong promise to greatly improve the quality of searches, so we will explore approaches that allow queries to include this information. Our language should allow queries to consider only particular links in establishing distance; this will allow queries to be restricted to files that are nearby or related from a particular user's point of view.

Problem: Finding Good or Correct Information is Difficult

Solution: Integrated, Scalable Indexing

Providing good performance and scalability for file search in an exascale system requires new indexing solutions. Existing file search systems rely on general-purpose, off-of-the-shelf solutions, such as relational database systems. However, these systems are typically designed for other workloads, such as business processing, and thus make few file search optimizations. As a result, they lack the performance and scalability to address file search at large scale. A file system search index must leverage file system and workload properties to achieve the necessary performance.

The approach we are proposing provides an index for two types of per-file metadata: simple metadata and content-based metadata. Searching against the different types of metadata uses a different process. Simple metadata can be compared using relatively static evaluations, similar to those provided in a relational database system: string equality, regular expressions, and numeric comparisons. However, content-based metadata is often evaluated for *similarity* to an exemplar; for example, a user might search for climate simulation results similar to the one she just produced. Similarity evaluation requires more computation,

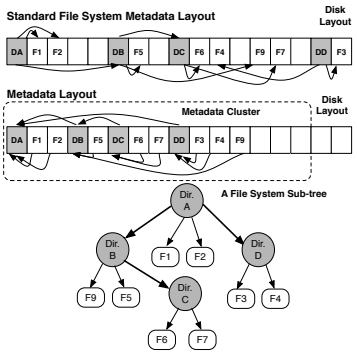

Figure 2: **Metadata clustering.** Each block corresponds to an i-node on disk. Shaded blocks labeled 'D' are directory i-nodes while non-shaded blocks labeled 'F' are file i-nodes. In the top disk layout, the indirection between directory and file i-nodes causes them to be scattered across the disk. The bottom disk layout shows how metadata clustering co-locates i-nodes for an entire sub-tree on disk to improve search performance.

and may also require application-specific routines to compute a similarity metric.

We will also expand on the functionality provided by file system indexes, providing features not typically available in current file systems and search indexes. There are several critical components to make search practical for everyday, real-world use. First, search must enforce file security, however, doing so efficiently is not straightforward. Our techniques allow security information to be used during index partitioning and embedded within each partition. Doing so allows us to eliminate partitions with improper permissions from the search space, improving performance and potentially altering the ordering of returned results. Second, search must allow the combination of per-file metadata with graph-based information to permit searches that find relevant "nearby" files. This functionality is not present in current Web-based searches: there is no way to find a page to which a given page links that contains a certain term, for example.

Problem: Veracity of Information is Often Unknown

Solution: Metadata Management

File system metadata should be treated as an aid to managing and accessing data and not a rigid and limited structure to which the user must conform. To this end we propose to enhance metadata management to provide seamless support for a search-based dynamic interface to the files. File system search provides a clean, powerful abstraction from the file system. It is often easier to specify *what* one wants using file metadata and extended attributes rather than specifying *where* to find it. Searchable metadata allows users and administrators to ask complex, ad hoc questions about the properties of the files being stored, helping them to locate, manage, and analyze their data.

We propose addressing this problem using metadata clustering, a concept similar to embedded i-nodes which store file i-nodes adjacent to their parent directory on disk. Metadata clustering goes a step further and stores a group of file i-nodes and directories adjacent on disk. Metadata clustering exploits several file system properties. First, disks are much better at sequential transfers than random accesses. Metadata clustering leverages this to pre-fetch an entire sub-tree in a single large sequential access. Second, file metadata exhibits *name space locality*: metadata attributes are dependent on their location in the name space. For example, files owned by a certain user are likely to be clustered in that user's home directory or their active project directories, not randomly scattered across the file system. Thus, queries will often need to search files and directories that are nearby in the name space. Clustering allows this metadata to be accessed more quickly using fewer I/O requests. Third, metadata clustering works well for many file system workloads, which exhibit similar locality in their workloads. Often, workloads access multiple, related directories, for which clustering works well.

Problem: Tracking Relations Among Documents

Solution: Provenance

Provenance information is a critical part of file system metadata information, since it provides insight into the processes that created a particular piece of data. Previous work has demonstrated that provenance information can be efficiently captured and stored for relatively small data collections; we will expand on this work to integrate provenance collection and management across exabyte-scale storage with billions of files. We will develop techniques for managing provenance at exabyte-scale along with other metadata, for guaranteeing integrity of provenance that records sufficient operations to recreate both the inputs and the workflow that generated a particular piece of data, and for efficiently extracting file attribute and inter-file relationship information from the file system.

We are developing new scalable techniques for extracting file metadata. Basic metadata—information currently stored in i-nodes and small-scale extended attributes—are easily handled by our approach, since updates can be sent directly to the MDS. Since they are relatively small, it is likely that they will only be updated by a single client even for a large file; thus, handling such updates requires relatively few MDS resources. However, file content in a HEC system is orders of magnitude larger than basic metadata, and extracting metadata from HEC files can be computationally and I/O intensive. Thus, we will rely upon *transducers* that are customized to each type of file. In our system, relationships between files are *first-class* entities, and can have tags and ownership information associated with them. Tracking such relationships requires monitoring calls to the file system (e.g. `open` and `close`) and providing additional API calls to explicitly establish relationships between files. In addition to providing API calls to explicitly link files, our system will implicitly gather provenance and other relationship information by monitoring processes on client nodes, and will coalesce that information either at the client level or on the MDS.