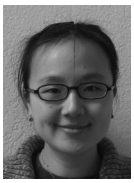
ANDREW W. LEUNG, MINGLONG SHAO, TIMOTHY BISSON, SHANKAR PASUPATHY, AND ETHAN L. MILLER

Spyglass: metadata search for large-scale storage systems

Andrew W. Leung is a PhD student at the University of California, Santa Cruz. His advisor is Ethan L. Miller. He is currently researching new techniques for large-scale data management, with a particular emphasis on storage system search. He received his BS from the University of California, Santa Barbara.

aleung@cs.ucsc.edu



Minglong Shao is a member of the Advanced Technology Group at NetApp. She received a Ph.D. in Computer Science from Carnegie Mellon University in 2007. Her research interests are in the area of database and storage systems, with an emphasis on database system behavior on modern storage devices.

minglong@netapp.com

Tim Bisson is a software engineer at NetApp in Sunnyvale, California. He received his Ph.D. in computer science from the University of California, Santa Cruz, in 2007.

tbisson@netapp.com

Shankar Pasupathy is a member of NetApp's advanced technology group, where he leads the indexing project. He is involved in research related to indexing petabyte-scale storage, as well as building large content repositories.

Shankar.Pasupathy@netapp.org

Ethan Miller is a professor of computer science at the University of California, Santa Cruz, where he is the Associate Director of the Storage Systems Research Center. His research interests include archival storage systems, scalable metadata management, file system reliability and security, scalable distributed storage systems, and file systems for non-volatile memories.

elm@cs.ucsc.edu

ARE OUR GROWING STORAGE SYSTEMS a blessing or a curse? As storage systems expand to billions of files they become increasingly difficult to manage. Effective management requires quickly searching the metadata of the files being stored. Unfortunately, tools such as find and grep and off-the-shelf databases cannot easily scale to billions of files. Spyglass is a new approach to metadata search, which leverages file and query properties to improve performance and functionality and to allow people to focus on using rather than merely managing their data.

The Growing Data Problem

Our ever-growing storage systems have become an escalating problem for storage users and administrators. Increasing amounts of digital data coupled with decreasing storage costs have yielded systems that store petabytes of data and billions of files. Managing this rising sea of data is difficult because users and administrators need to efficiently answer a variety of questions about the files being stored. For example, a user may need files that he/she has forgotten the location for (e.g., "Where are my recently modified presentation files?"), or an administrator may need to reduce storage capacity (e.g., "Which files can be migrated to second-tier storage?"). These questions are extremely difficult to answer when one must sift through billions of files. Moreover, answering data management questions requires complex query functionality. Consider a scenario where a buggy script accidentally deletes a number of users' files. To fix the problem, an administrator would like to answer the question, "Which files should be restored from backup?" Restoring the entire backup takes enormous amounts of time and destroys any changes made since then. Ideally, an administrator would like to be able to search both the current and previous file versions, search for just those files affected by the script, and determine which critical files should be restored first.

Metadata search can greatly aid users and administrators in answering these questions. Metadata search allows ad hoc queries over file properties. These properties are structured *attribute,value* pairs and consist of metadata such as inode fields (e.g., size, owner, timestamps) and extended attributes

(e.g., document title, retention policy, backup dates). Metadata search helps users and administrators understand the kinds of files being stored, where they are located, how they got there (provenance), and where they belong, and is a key step towards improving how we manage our data. Table 1 shows examples of some popular metadata search queries from a survey we conducted, which we detail later in the article.

File Management Question	Metadata Search Query
Which files can I migrate to tape?	size > 50 GB, atime > 6 months
How many duplicates of this file are in my home directory?	owner = john, datahash = 0xE431, path = /home/john
Where are my recently modified presentations?	owner = john, type = (ppt | keynote), mtime < 2 days
Which legal compliance files can be expired?	retention time = expired, mtime > 7 years
Which of my files grew the most in the past week?	Top 100 where size(today) > size (1 week ago), owner = john.
How much storage do these users and applications consume?	Sum size where owner = john, type = database

TABLE 1: USE-CASE EXAMPLES. METADATA SEARCH USE-CASES COLLECTED FROM OUR USER SURVEY. THE HIGH-LEVEL QUESTIONS BEING ADDRESSED ARE ON THE LEFT. ON THE RIGHT ARE THE METADATA ATTRIBUTES BEING SEARCHED AND EXAMPLE VALUES.

Metadata search is becoming increasingly popular and is common on desktop and small-scale enterprise storage systems. Most desktop file systems ship with a search tool that supports metadata search [1, 11]. Enterprise search appliances, such as FAST [4] and Google Enterprise [5], provide metadata search for relatively small-scale storage systems (e.g., up to tens of millions of files). These tools are becoming more prevalent; recent studies show that 37% of enterprise businesses already use such tools and that 40% plan to use them in the near future [6].

Unfortunately, it appears that we are still not ready to address metadata search at large scales. Searching storage systems with billions of files gives rise to a number of challenges that existing solutions do not address. First, cost and resources must be efficiently utilized. Existing enterprise-scale solutions address performance through dedicated CPU, memory, and disk hardware, which quickly becomes prohibitively expensive at large scales. It can cost tens of thousands of dollars just to search tens of millions of files [8]. An effective solution should be able to reside directly within the storage system, sharing resources without degrading search or native storage performance.

Second, there must be a way to quickly gather metadata changes. Large systems can produce millions of metadata changes per minute. However, existing solutions often gather metadata changes with a brute-force crawl of the storage system, which is not only prohibitively slow but also extremely resource-intensive. We have observed commercial systems take 22 hours to crawl 500GB and 10 days to crawl 10TB. Other approaches, such as intercepting file system requests, are often not possible because enterprise appliances are not integrated with the storage system.

Third, search and update performance must be highly scalable. It is very difficult to search billions of files in a timely manner. Poor search and update performance directly impacts overall effectiveness and usability. Existing

solutions rely on basic off-the-shelf solutions—in most cases, general-purpose relational databases (DBMSes). While privy to decades of performance research, DBMSes are not designed or optimized for storage system search. As a result, they make few metadata search optimizations and have extraneous functionality that adds overhead. Our results show that a simple DBMS-based solution often requires many minutes and sometimes hours to search tens to hundreds of millions of files. The concept that a general-purpose DBMS should not serve as a "one size fits all" solution is not new and is actually touted by the DBMS community itself [12].

Re-Thinking Metadata Search

Addressing metadata search requires rethinking our approach to the problem, since we cannot rely on basic off-the-shelf solutions. As with any good system design, the key to improving performance and scalability is to understand and leverage metadata search properties. This requires understanding the properties of the files being searched and the queries being run. To do this we surveyed over 30 storage users and administrators from NetApp and the University of California, Santa Cruz, and analyzed metadata from three storage systems at NetApp. Armed with recent observations, we developed a new metadata search system, called Spyglass, which was presented at the 7th USENIX Conference on File and Storage Technologies (FAST '09) [9]. Spyglass introduces new approaches to index design, index updating, and metadata collection that leverage metadata search properties to improve performance and scalability.

We asked our survey participants, "What kinds of metadata queries would you like to perform?" Our survey yielded interesting insights, with some of the popular searches featured in Table 1. Queries almost always contain multiple metadata attributes, because querying a single attribute returns too many results to be useful. Also, queries can often be localized to only a part of the namespace, such as a home or project directory, by using a pathname in the query. Often participants had some idea where their data was located, and localizing queries helps to narrow the results. Finally, queries often involve "back-in-time" search of previous metadata versions to answer questions about how or when files have changed. Spyglass leverages this information by using a query execution method that utilizes all of the attributes in the query, embedding namespace information directly into the index, and versioning index updates.

The first storage system we analyzed served Web server data and stored about 15 million files. The second hosted engineering build space (source, object, and support files) and stored about 60 million files. The third stored employee home directories and contained about 300 million files. All three systems exhibit two important metadata characteristics: metadata values are highly clustered in the namespace, and the distribution of metadata values greatly changes when looking at a single metadata value versus a combination of values. Metadata clustering means that any particular metadata value, such as *owner,Andrew*, occurs only in a tiny fraction of directories (e.g., *Andrew*'s home directory) as opposed to being scattered across the namespace. In fact, most of the values we studied occurred in less than one-tenth of 1% of the directories! This is not a surprising result, really, since people use the namespace to group related files. Spyglass leverages this information by identifying just the few namespace locations that must be searched for in a query, thereby greatly reducing the scope of the search. Existing solutions do not utilize namespace information and must consider files from the entire storage system.

Our other observation is that the distributions of single metadata values differ greatly from those of multiple metadata values. Consider an example from our engineering storage system. As one might expect, there are many millions of *.c* source files, which comprise a large fraction of all file types on the system (we found that these distributions closely followed the power-law distribution). However, when we consider multiple attribute values at once, such as *file type*,*.c* and *owner*,*Andrew*, there are far fewer files with *both* values (we often found orders-of-magnitude fewer files). Spyglass leverages this knowledge by using *all* values when executing a query, which greatly reduces the number of files that must be considered. Existing DBMS-based solutions often rely on only a single attribute value when executing queries, and performance often suffers when distributions are skewed [10].

The Spyglass Design

Spyglass is designed to reside directly within the storage system and consists of two main components, shown in Figure 1: the Spyglass index, which stores metadata and serves queries, and a crawler that extracts metadata from the storage system.

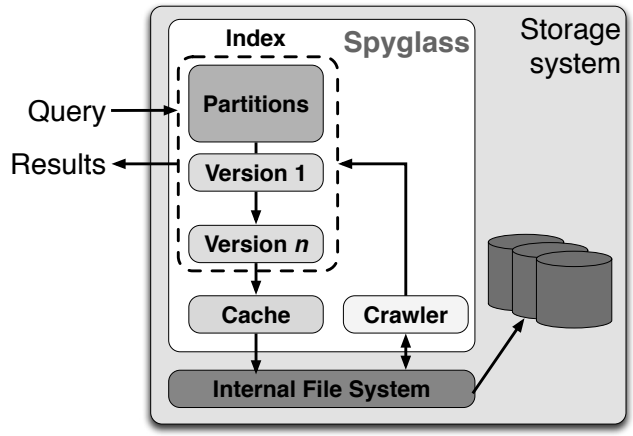


FIGURE 1: SPYGLASS OVERVIEW. SPYGLASS RESIDES WITHIN THE STORAGE SYSTEM. THE CRAWLER EXTRACTS FILE METADATA, WHICH GETS STORED IN THE INDEX. THE INDEX CONSISTS OF A NUMBER OF PARTITIONS AND VERSIONS, ALL OF WHICH ARE MANAGED BY A CACHING SYSTEM.

Spyglass introduces several new metadata search designs. First, *hierarchical partitioning* partitions the index based on the namespace, allowing the index to exploit metadata clustering and allowing fine-grained index control. Second, *signature files* [3] are used to automatically identify the partitions that are relevant to a query. Third, *partition versioning* versions index updates, which improves update performance and allows "back-in-time" metadata search. Finally, Spyglass utilizes snapshots in WAFL [7], the file system on which it was built, to collect metadata changes by crawling only files whose metadata has changed. In this article we briefly describe the design of the Spyglass index and how it uses hierarchical partitioning and signature files. A complete description of the design can be found in our FAST '09 paper [9].

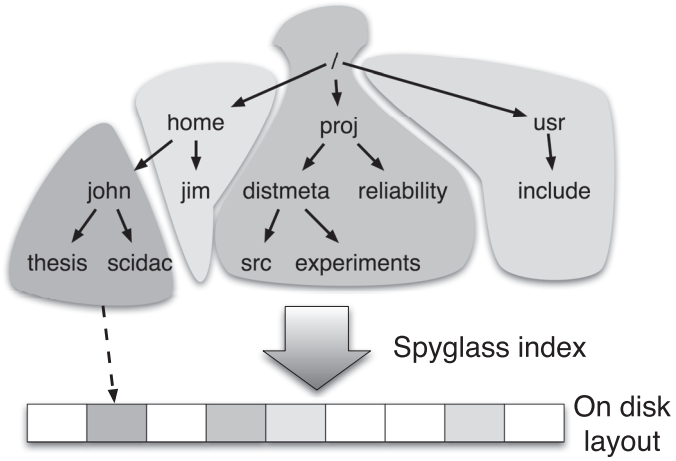


FIGURE 2: HIERARCHICAL PARTITIONING EXAMPLE. SUB-TREE PARTITIONS, SHOWN IN DIFFERENT COLORS, INDEX DIFFERENT STORAGE SYSTEM SUB-TREES. EACH PARTITION IS STORED SEQUENTIALLY ON DISK. THE SPYGLASS INDEX IS A TREE OF SUB-TREE PARTITIONS.

Hierarchical partitioning indexes files from separate parts of the namespace into separate partitions, providing flexible, fine-grained index control at the granularity of sub-trees. Figure 2 shows an example where files from separate sub-trees are indexed in separate partitions. The key idea is that we can now search only the parts of the storage system that are relevant to our query, without concerning ourselves with files from other parts of the system. Our finding that metadata values are highly clustered in the namespace indicates that in most cases we can search only a small fraction of the partitions. Spyglass keeps partitions relatively small (on the order of 100,000 files) and stores them sequentially on disk. This layout ensures very fast access to any one partition. Each partition indexes its metadata in a K-D tree [2].

Spyglass search performance is a function of the number of partitions that must be searched. Thus, when executing a query the question becomes, "Which partitions must be searched?" Spyglass can identify partitions in two ways. First, users can localize their queries (which we found to be common in our survey) and thereby control the number of partitions searched. Spyglass also uses signature files [3] to determine which partitions *may* have files relevant to a query. A signature file is a bit array with an associated hashing function that is a compact representation of a partition's contents. Each partition has a signature file for each attribute that it indexes, which are kept small so that they may fit in memory. Metadata values that are indexed in the partition have associated signature file bits set to one. If and only if *all* metadata values in a query map to one bits in the signature files is a partition read from disk and searched. Spyglass leverages the fact that there are far fewer files matching all query values than if only a single value is used. Since signature files are probabilistic, false positives can occur.

An evaluation of our Spyglass prototype shows significant performance improvements compared to a simple DBMS-based solution. Our evaluation was done using the real-world metadata we collected and sets of queries derived from our survey. We evaluated the performance of two popular DBMSes, PostgreSQL and MySQL, to serve as relative comparison points to DBMS-based solutions used in other metadata search systems. We found that Spyglass satisfies most queries in less than a second even as the system scales

from 15 to 300 million files. Few queries took less than a second with our reference DBMSes. In fact, many queries require well over five minutes on the largest data set. A key reason for the performance difference is that Spyglass leverages hierarchical partitioning to greatly reduce the overall search space and often only requires a few small, sequential disk reads to answer a query. We also found that localizing a query to only a part of the namespace enhances performance to the point where Spyglass is up to four orders of magnitude faster than our reference DBMSes in some cases.

The Role of Search in the Storage System

Spyglass takes a first step towards improving how we manage our growing storage systems through new metadata search designs that leverage storage system properties. However, easily and effectively managing billions of files remains a daunting task. A key reason for this is that we must rethink the relationship between search and the file system. While becoming more common, file search (Spyglass included) remains an application that is often completely distinct from the file system. Yet search and file systems share a common goal: organizing and retrieving file data. Keeping search completely separate from the file system leads to duplicate functionality (e.g., each stores and maintains separate file index structures) and contention for resources (e.g.,a file requires memory and disk resources in both the file system index and search index). It's not hard to imagine the problems that will arise in the future when trying to index hundreds of billions of files in *both* the file system and a search index. Additionally, users are forced to interact with multiple interfaces (e.g.,the POSIX file system interface and the search interface), which needlessly complicates interactions with the storage system.

We believe improving the relationship between file search and the file system is key to improving how we manage our growing storage systems. We plan to explore this issue in our on-going Spyglass work. Our main goal is to explore how search can be integrated within the file system as first-class functionality. From an abstract level, Spyglass closely resembles a file system in which directory and file metadata are embedded together in partitions. Any file's metadata can be accessed in Spyglass by hierarchically traversing the partitions until the partition containing the file is reached, similar to a file system. The key differences from a file system are that in Spyglass files are grouped on disk by partitions rather than directories and that Spyglass versions each partition's file modifications (discussed in our FAST '09 paper) rather than perform in-place updates. Our on-going work looks at how an architecture like Spyglass may be used as the *main* metadata store for a storage system. Using an approach like Spyglass for all metadata storage means that only a single data structure, the Spyglass index, needs to be maintained, updated, cached, and searched. Doing so has the potential to improve overall performance by more efficiently utilizing resources and can greatly improve the functionality offered by the file system. The key research challenges will be achieving good performance for more general metadata workloads and achieving efficient real-time index updates.

ACKNOWLEDGMENTS

We would like to thank our colleagues in the Storage Systems Research Center and NetApp's Advanced Technology Group for their input and guidance. We also owe thanks to Remzi Arpaci-Dusseau, Stavros Harizopoulos, Jiri Schindler, and our FAST '09 shepherd, Sameer Ajmani, and the FAST '09 reviewers whose comments significantly improved this work.

This work was supported in part by the Department of Energy's Petascale Data Storage Institute under award DE-FC02-06ER25768 and by the National Science Foundation under award CCF-0621463. We thank the industrial affiliates of the SSRC for their support.

REFERENCES

[1] Apple, "Spotlight Server: Stop Searching, Start Finding": http://www.apple.com/server/macosx/features/spotlight.html, 2008.

[2] J.L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM,* 18(9): 509–517 (1975).

[3] C. Faloutsos and S. Christodoulakis, "Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation," *ACM Transactions on Information Systems,* 2(4): 267–288 (1984).

[4] FAST—Enterprise Search: http://www.fastsearch.com/, 2008.

[5] Google Enterprise: http://www.google.com/enterprise/, 2008.

[6] E.S. Groups, "ESG Research Report: Storage Resource Management on the Launch Pad," 2007.

[7] D. Hitz, J. Lau, and M. Malcom, "File System Design for an NFS File Server Appliance," *Proceedings of the Winter 1994 USENIX Technical Conference*, USENIX Association, 1994, pp. 235–246.

[8] Goebel Group, Inc., "Compare Search Appliance Tools": http://www.goebelgroup.com/sam.htm, 2008.

[9] A. Leung, M. Shao, T. Bisson, S. Pasupathy, and E.L. Miller, "Spyglass: Fast, Scalable Metadata Search for Large-Scale Storage Systems," *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST '09)*, USENIX Association, 2009, pp. 153–166.

[10] C.A. Lynch, "Selectivity Estimation and Query Optimization in Large Databases with Highly Skewed Distribution of Column Values," *Proceedings of the 14th Conference on Very Large Databases (VLDB)*, 1988, pp. 240–251.

[11] Microsoft Windows Search 4.0: http://www.desktop.google.com/, 2007.

[12] M. Stonebraker and U. Cetintemel, "'One Size Fits All': An Idea Whose Time Has Come and Gone," *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, 2005, pp. 2–11.