Journal of Physics: Conference Series **125** (2008) 012069 doi:10.1088/1742-6596/125/1/012069

High-performance metadata indexing and search in petascale data storage systems

A W Leung*, M Shao†, T Bisson†, S Pasupathy†and E L Miller*

*Storage Systems Research Center, University of California, Santa Cruz, CA 95064, USA
†NetApp, Inc., Sunnyvale, CA 94089, USA

E-mail: {aleung,elm}@cs.ucsc.edu, {minglong,tbisson,shankarp}@netapp.com

Abstract. Large-scale storage systems used for scientific applications can store petabytes of data and billions of files, making the organization and management of data in these systems a difficult, time-consuming task. The ability to search file metadata in a storage system can address this problem by allowing scientists to quickly navigate experiment data and code while allowing storage administrators to gather the information they need to properly manage the system. In this paper, we present Spyglass, a file metadata search system that achieves scalability by exploiting storage system properties, providing the scalability that existing file metadata search tools lack. In doing so, Spyglass can achieve search performance up to several thousand times faster than existing database solutions. We show that Spyglass enables important functionality that can aid data management for scientists and storage administrators.

1. Introduction

Recent years have seen an enormous increase in the amount of data placed on storage systems. This is particularly true in the scientific community where research, development, and experimentation have become largely digital processes. Scientific applications demand both scale and performance from their data storage systems. These applications produce petabytes of data spread across billions of files, and often have running times tied directly to how quickly they can access data. Building large scale, high performance storage systems required by scientific applications has been a major area of research for over two decades [1, 2, 3, 4, 5]; however, while these systems can provide the necessary performance, the scale of these systems has given rise to a new challenge: managing and organizing petabytes of data and billions of files.

The mere presence of billions of files and petabytes of data makes it difficult for users to find and organize their data and complicates management decisions for storage administrators, particularly in scientific environments. An experiment can produce millions of files that must be analyzed to produce useful results, and both users and administrators must manage the files from many experiments. For scientists, this situation can waste time and can cause important results to be missed. Administrators face more difficult decisions, potentially jeopardizing data on the storage system.

A significant portion of the data management problem can be addressed through the ability to rapidly search file metadata in a storage system; the importance of search in a storage system is comparable to its importance on the Web. Large-scale systems consist of a sea of loosely organized data identified by a location (path name in a storage system and URL on the web) that must be known in order to access it. Search allows users to use metadata about the file

to locate it without requiring the path to be known, a technique far superior to the manual browsing and organizing done today. In a storage system, metadata search allows a scientist to quickly search for relevant experimental results, and it provides a means for administrators to gain important information about the data they manage.

Recently, tools such as Apple's Spotlight and Google Desktop have become available to users of desktop file systems. While these tools are effective in small-scale file systems, they lack the scalability to address file metadata search in large-scale storage systems. because they do not make any significant optimizations or designs for file search. Instead, they rely upon traditional indexing techniques, such as relational database systems, that are designed for very different workloads, resulting in systems that are often too slow to search and too slow to update and require too many system resources, such as memory and disk space.

In this paper, we discuss the design and implementation of scalable systems for searching and indexing file metadata that are capable of addressing the search needs of scientists and their storage administrators. We first present background information for search in large-scale scientific computing. We then discuss Spyglass [6], a system we have designed for searching large-scale storage systems that is up to several thousand times faster than existing database solutions. Next, we provide examples of how Spyglass can aid scientific research. Finally, we discuss related and future work and then conclude.

2. Background

File metadata search works by building an index of fields in a file's inode, such as owner and size, and its extended attributes, which are manually defined key-value pairs. An example of an extended attribute key-value pair may be a key (e.g., `experiment_runtime`) and value (e.g., `2 hours`). An example metadata query might ask, "Which experiments did I run last week that took less than 20 minutes to run?" An index is a data structure that allows the rapid identification of files that contain specific metadata values. User queries can quickly be satisfied by simply looking up the information in the index. Without an index, the file system must be manually traversed, reading each file's metadata, in order to satisfy a query, a process far too slow to be effective.

Many existing systems, such as Apple's Spotlight and Linux's Tracker, index metadata using relational database systems (DBMS). DBMSs build relational tables in which each column in a table defines a different metadata field and each row defines a different file's metadata values. In general, queries are satisfied by searching the table for files that match a query's predicates. In large-scale storage systems, however, scanning a table can be very slow because the table is so large. Thus, B+-tree indexes are often built on the columns that need fast search. A B+-tree is a data structure that allows the files with specific values for a key to be quickly located through logarithmic search of the key's sorted value space, eliminating much of the needless searching required by a table scan.

However, a DBMS does not make any optimizations for storage system workloads; instead, the design is based on data warehousing workloads from the 1970s. As a result, a generic DBMS is not a "one size fits all solution" for indexing data, a sentiment echoed by the DBMS community itself [7]. File systems have common access patterns [8, 9] that can be exploited to improve performance. For example, requests are not evenly distributed across storage namespace locations, meaning some subtrees and directories are more popular than others. Also, requests have locality of reference: if a subtree of a directory was recently accessed, it is likely to be accessed again soon. Additionally, metadata values tend to be clustered in small parts of the file system namespace. For example, most files owned by a user are located in their home directory, with others in specific project directories in which the user is actively working. By ignoring many such properties of file systems, DBMSs have a number of inefficiencies that limit their scalability.

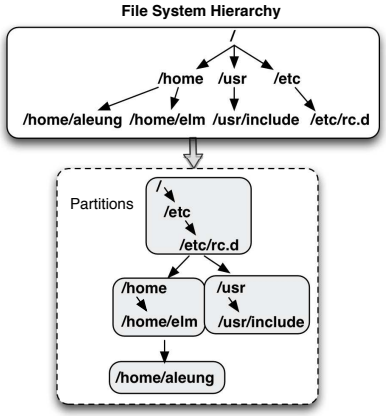

Figure 1. Hierarchical Partitioning. Spyglass partitions the index based on the storage system's namespace. This allows search and indexing to be controlled at the finer granularity of subtrees. We can exploit the namespace's locality and reference properties to reduce a query's search space and improve performance.

3. Spyglass design

Our metadata indexing system, Spyglass, introduces two index design methods that are optimized for large-scale storage systems: *hierarchical partitioning* and *partition versioning*. Hierarchical partitioning takes advantage of the storage system's locality properties by partitioning the index into a number of smaller partitions, based on the file system namespace, with each partition responsible for indexing one or more subtrees in the namespace. An example of this partitioning is shown in figure 1, in which each partition indexes files from different subtrees in the namespace. The key idea is that hierarchical partitioning allows search and indexing to be controlled at the granularity of subtrees, which cannot be done when all data is stored in a single index. Using this, we can take advantage of many storage system workload properties. For example, metadata attributes tend to be clustered in the storage namespace, meaning that, in most cases, few partitions are required to satisfy a query, greatly reducing a query's search space. Similarly, some parts of the namespace are frequently accessed, so caching the corresponding index partitions in memory can greatly improve performance for common-case queries.

Partition versioning simplifies operation and improves performance of index updates and allows users to search over past file metadata. Spyglass does this by batching changes made to file metadata over an interval (the length of the interval is a file system parameter) and then applying each batch to the index as a new index version. Thus, when the index is updated, a new version does not overwrite old an old version but instead allows old metadata to be maintained and searched by the user. This feature is important for management and trend queries where metadata information overtime is needed. To save space, each version stores only metadata changes since the previous version, rather than storing a full copy of all file metadata in the system; thus, partition versions are incremental. Also, each partition manages its own versions, allowing partition versioning to take advantage of storage workload properties. For example, because only a few popular namespace locations frequently change, it is usually the case that only a few partitions must be versioned. Likewise, versions for popular partitions can be kept in cache with the partition itself, meaning most queries over multiple versions are very fast.

These optimizations to file system indexes both reduce resource usage and improve performance. Because each index partition covers only a small part of the file system namespace, each partition can be individually "customized" to the metadata it contains. For example, while

there may be thousands of users who have created files in the file system, an individual index partition may only contain files owned by fewer than ten users, allowing the index to dramatically compress the representation of the users' ID. The index can perform similar compression on other attributes, including timestamps, permissions, file types, and other extended attributes.

Spyglass further improves performance by using *signatures* to quickly identify partitions that may be relevant for a particular query. For example, each index partition lists the users that own files it indexes. If a query requests files owned by user A, only those files that contain user A's files are even considered by the query program. For the localized queries common in file system usage, this technique quickly rules out most of the index partitions, leaving only the partitions that might contain positive results for the query.

To demonstrate the advantages of using the partitioned, versioned indexes in Spyglass, we conducted a number of experiments that compare Spyglass performance with existing DBMS solutions using real-world large-scale storage system metadata traces. We found that Spyglass requires between 5 to 8 times less disk space and updates between 8 to 44 times faster than DBMS systems. The reason is that DBMSs use a B+-tree for each metadata attribute that needs fast query performance, each of which adds space and update overhead. Spyglass indexes each metadata value just once by using multi-dimensional data structures. We also found that Spyglass query performance was at least 10 times faster than DBMSs and up to several thousand times faster in some cases, particularly those in which users were able to localize their search to only a part of the storage namespace. Because hierarchical partitioning allows control over the granularity of the search, a query's search space can be significantly reduced, reducing the total amount of index data that must be processed for the query.

4. Example applications

Fast metadata search enables scientists and large-scale storage administrators to improve their efficiency and better do their jobs. Suppose a scientist runs a long experiment that produces a million files. Using existing file system interfaces, such as `xattr_set()`, extended attributes about the experiment can be set and indexed in Spyglass. A scientist may want to create extended attributes that describe whether the experiment was successful, how long it took, or the results contained in the files. Without any method to search these files, the scientist must manually navigate the files, a slow and imprecise process, or use an external index to manage the files, an approach that cannot leverage tight integration with the file system. Using Spyglass, however, the scientist can simply search for the desired files using queries such as "Which experiment files that I created yesterday were successful and finished in less than 1 hour?" Additionally, by using the versioning abilities of Spyglass, the scientist can ask trend queries about experiments. Using a chemical experiment as an example, a scientist could ask how the pH levels in particular acidity experiments been changing over the past week, a question that is difficult, if not impossible, to answer with existing tools.

Storage administrator who manage a storage system for scientific data can use Spyglass to aid their management decisions. For example, the administrator can easily find which old, unsuccessful experiments to delete by asking, "Which unsuccessful experiment files have not been accessed in over a month?" Additionally, the administrator can improve data security by searching for and encrypting all files that have "nuclear bomb test" attributes.

5. Future work

Scalable file search in large-scale storage systems still presents a number of unsolved challenges. We plan to extend our existing work on metadata indexing in Spyglass to several of these other areas. We are planning on exploring techniques that apply hierarchical partitioning to improve the scalability of content-based search. Content-based search allows keywords to be automatically extracted from files for indexing and differs from metadata search in the number

of possible keywords that can be searched. Extracting keywords efficiently is also a challenge, especially for large scientific files, which we also plan to explore in the future. Given that many high-performance storage systems are distributed, indexing solutions for these system must also be distributed. Thus, we also plan to look at scalable communication protocols for index consistency and query execution.

6. Conclusions

The scientific communities need for scalable storage, high-performance storage has yielded storage systems capable of storing petabytes of data across billions of files. However, the scale of these systems has made managing data on them a difficult task. The ability to search file metadata in a storage system can greatly ease the burden on scientists and storage administrators. However, existing file search tools lack the scalability effectively search large-scale storage systems, largely because they are optimized for storage systems.

In this paper, we presented Spyglass, a scalable metadata search and indexing system that is designed specifically for storage systems. By designing using an index that exploit storage properties, Spyglass is able to achieve significantly better query performance than existing DBMS solutions, while using less disk space. We showed through example how its functionality can be used to aid the data management in the scientific community.

Acknowledgments

This work was supported in part by the Department of Energy under the Petascale Data Storage Institute (award DE-FC02-06ER25768) and industrial sponsors of the Storage Systems Research Center at UC Santa Cruz, including Agami Systems, Data Domain, Hewlett Packard, Hitachi, LSI, NetApp, Seagate, and Symantec. We also thank our colleagues in the SSRC and NetApp's Advanced Technology Group for their insightful feedback, which greatly improved the quality of the research.

References

[1] Corbett P F and Feitelson D G 1996 The Vesta parallel file system *ACM Trans on Computer Systems* **14** 225–264

[2] Gibson G A, Nagle D F, Amiri K, Butler J, Chang F W, Gobioff H, Hardin C, Riedel E, Rochberg D and Zelenka J 1998 A cost-effective, high-bandwidth storage architecture *Proc. 8th Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (San Jose, CA) 92–103

[3] Miller E L and Katz R H 1997 RAMA: an easy-to-use, high-performance parallel file system *Parallel Computing* **23** 419–446

[4] Schmuck F and Haskin R 2002 GPFS: a shared-disk file system for large computing clusters *Proc. 1st Conference on File and Storage Technologies (FAST)* 231–244

[5] Weil S A, Brandt S A, Miller E L, Long D D E and Maltzahn C 2006 Ceph: a scalable, high-performance distributed file system *Proc. Symp. Operating Systems Design and Implementation (OSDI)*

[6] Leung A W, Shao M, Bisson T, Pasupathy S and Miller E L 2008 Spyglass: fast, scalable metadata search for large-scale storage systems Tech. Rep. UCSC-SSRC-08-01 Storage Systems Research Center, University of California Santa Cruz

[7] Stonebraker M and Cetintemel U 2005 "One size fits all": an idea whose time has come and gone *Proc. 21st Intl. Conference on Data Engineering* (Tokyo, Japan) 2–11

[8] Leung A W, Pasupathy S, Goodson G and Miller E L 2008 ' Measurement and analysis of large-scale network file system workloads *Proc. USENIX Annual Technical Conference*

[9] Roselli D, Lorch J and Anderson T 2000 A comparison of file system workloads *Proc. USENIX Annual Technical Conference* 41–54