Security Aware Partitioning for Efficient File System Search

Aleatha Parker-Wood, Christina Strong, Ethan L. Miller, Darrell D.E. Long
Storage Systems Research Center
University of California, Santa Cruz
$\{$*aleatha, crstrong, elm, darrell*$\}$*@cs.ucsc.edu*

Abstract

Index partitioning techniques—where indexes are broken into multiple distinct sub-indexes—are a proven way to improve metadata search speeds and scalability for large file systems, permitting early triage of the file system. A partitioned metadata index can rule out irrelevant files and quickly focus on files that are more likely to match the search criteria. Also, in a large file system that contains many users, a user's search should not include confidential files the user doesn't have permission to view. To meet these two parallel goals, we propose a new partitioning algorithm, Security Aware Partitioning*, that integrates security with the partitioning method to enable efficient and secure file system search.*

In order to evaluate our claim of improved efficiency, we compare the results of Security Aware Partitioning to six other partitioning methods, including implementations of the metadata partitioning algorithms of SmartStore and Spyglass, two recent systems doing partitioned search in similar environments. We propose a general set of criteria for comparing partitioning algorithms, and use them to evaluate the partitioning algorithms. Our results show that Security Aware Partitioning can provide excellent search performance at a low computational cost to build indexes, $O(n)$. Based on metrics such as information gain, we also conclude that expensive clustering algorithms do not offer enough benefit to make them worth the additional cost in time and memory.

1. Introduction

From a consumer's standpoint, storage is cheap. Individuals have personal computers with external storage; companies, scientific institutions, and academia all garner benefits from file sharing and shared backup by storing data on petabyte scale file systems—or larger—with hundreds or even thousands of users. With the advent of cloud computing, individuals also may opt to store and share their personal files in exabyte scale file systems accessible via the Internet. In shared file systems, users need their personal data to remain private and not show up as a result in an unauthorized user's search. This is particularly crucial in a corporate setting. Confidential information often has severe legal and financial consequences if leaked, ranging anywhere from a fine for a violation of the U.S. Securities and Exchange Commission (SEC) regulations [6] or the Health Insurance Portability and Accountability Act (HIPAA) [5] to the loss of consumer trust when confidential user data—such as credit card information or social security numbers—is released. Similarly, scientific and academic institutions maintain a level of confidentiality surrounding their work. While the consequences are not necessarily as far-reaching, no scientist wants to find that someone else published the results he was collecting.

With both the size of file systems and the number of files stored increasing, it becomes increasingly important for file systems to offer fast scalable search. What is more, individuals have come to expect the high quality split second results that popular web ranking algorithms [11], [22] provide. A file system's hierarchical structure provides different information than the highly connected graph of the web; it is these connections in the web that ranking algorithms exploit for fast results. While some file systems have attempted to simulate the web's structure [7], current file system search is fundamentally different from a standard web search. File systems contain huge amounts of rich metadata in a meaningful hierarchy, as well as a complex security model that has no web analogue. The ability to query over metadata as well as content is key to good file system search, and a successful search algorithm will be one which exploits the properties specific to file systems as well as respecting its security restrictions.

978-1-4244-7153-9/10/$26.00 © 2010 IEEE

Partitioning algorithms are a proven way to improve metadata search speeds for large file systems [24], [21], when a monolithic metadata index is too large to fit comfortably into main memory. In such systems, the file system is divided into multiple pieces, each of which has its own metadata index stored sequentially on disk. The contents of each index are represented by a series of Bloom filters [10]—a probabilistic form of a signature file—generally one for each type of metadata the system can search over, as illustrated in Figure 1. These Bloom filters can be quickly compared to the user's query before deciding whether to load a full index into memory. However, partitioning is ineffective and potentially worse than a monolithic index if every index must be searched. Ergo, an efficient partitioning strategy is one in which partitions are small enough to fit into main memory and only a few partitions must be searched to find the results of a query.

We propose a new partitioning algorithm, *Security Aware Partitioning*, which is advantageous for file systems with a variety of users and security permissions. While other systems have explored the question of security, it is usually at the cost of performance. Wumpus [12], for example, requires a high overhead security manager and results in a decreased search performance while the access checking solution of Bailey et al. [9] decreases the match count accuracy. Security Aware Partitioning takes the security permissions of the file into account while *creating* partitions, such that when a user queries the file system, only files which the user has permission to access are considered for the results. By building these restrictions into the partitions, it increases the search performance at query times while decreasing the security risk.

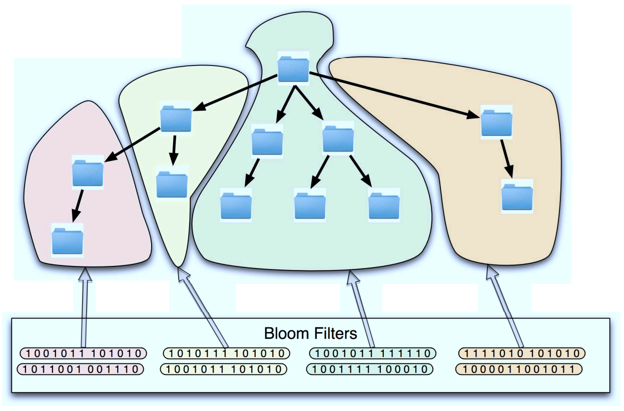


Figure 1. Partitioning divides the system into multiple regions, each with its own metadata index. Bloom filters are used to quickly triage partitions without relevant documents. In this diagram, hierarchy has been used as the partitioning criterion.

Our claim of increased search performance is not unique: many other systems, such as Spyglass [24] and SmartStore [21], have proposed different methods of partitioning large file systems, each promising improved search performance. However, these algorithms range widely in computational complexity. It is unclear whether complex algorithms for partitioning are truly beneficial, or if similar search performance can be achieved using simpler algorithms.

To this end, in the evaluation of our Security Aware Partitioning algorithm we also examine other partitioning schemes. We compare the results to determine whether simple algorithms can generate partitions that are comparable to more sophisticated algorithms for a lower cost at indexing time. We examine the quality of generated partitions using self-similarity and use information gain to give an estimation of efficiency at query time for searching the system. Finally, based on our findings, we discuss selection criteria for a partition scheme appropriate to a given environment.

The rest of the paper is structured as follows: We begin by presenting the Security Aware Partition algorithm in Section 2. In Section 3 we describe our experimental setup and the measures we used to quantify partitioning quality. We discuss our results and their implications in Section 4, and explore previous work in partitioning and security for search in Section 5. We then look forward to the future of file systems and what this means for partitioned search indexes in Section 6 and the future of this research in Section 7. Finally, we conclude in Section 8 with a summary of our findings.

2. Algorithms

While most partitioning algorithms, including our own, claim an advantage over others in some manner, the results are presented as compared to database algorithms used as a baseline. We chose instead to evaluate a number of partitioning algorithms to substantiate our claim of improved efficiency. In addition to evaluating our Security Aware Partitioning algorithm, we also evaluate the following six algorithms.

- A greedy time based algorithm
- An interval time based algorithm
- User based partitioning
- Cosine correlation clustering
- Cosine correlation clustering with Latent Semantic Analysis (LSA) – SmartStore [21]
- Greedy depth first search partitioning – Spyglass [24]

We present the motivation behind Security Aware Partitioning and the logistics of how it works. We then give an overview of the other partitioning algorithms and our motivation for choosing them.

2.1. Security Aware Partitioning

Security Aware Partitioning is a new algorithm we developed to support fast, scalable search, while maintaining the security of files. Since not every user has access to every file, displaying search results that are not accessible to a user is both insecure and poor user interface design. A list of results should only contain documents which are accessible to the user initiating the search.

Some partitioning schemes apply a filtering operation after the search results have been collected in order to enforce security restrictions. Bailey et al. [9] found that such schemes require a query time that grows linearly with respect to the number of potential matches. In order to reduce the overhead caused by such approaches, Security Aware Partitioning takes these requirements into account when building the partitions. This increases search efficiency and can prevent statistical attacks on ranked search, such as the attack demonstrated by Büttcher [12]. In our partitioning scheme, if someone has permission to access one file in a partition, he can access every file in that partition. However, to determine what "permission to access" means, it is necessary to briefly look at the security model of the underlying file system.

2.1.1. File Permissions. Each operating system implements file permissions slightly differently, but there are two standard models: *Access control lists (ACLs)* and "traditional" UNIX permissions. ACLs, most commonly used by NTFS file systems, associate an object (such as a file) with who may access the object and in what way. An ACL is comprised of zero or more entries specifying permissions for a given operation for various users and groups. For a requested operation, the system looks at each entry in the ACL until one of three things happens: an access-denied entry prohibits access, one or more access-allow entries explicitly allow access, or there are no more entries. A request that has not been explicitly allowed is denied access.

Traditional UNIX permissions are used by UNIX-like and other POSIX compliant operating systems, including Linux and Mac OS X. Traditional UNIX permissions use a nine bit permission model, where every three bits represent *read*, *write*, or *execute* permission for each of three security levels: the file's assigned *user*, assigned *group*, and *other*, defined as all users who do not fall into the first two categories.

In practice, while ACLs allow a more complex and detailed set of permissions, most are used to implement the same permissions as traditional UNIX permissions: *read*, *write*, or *execute*. In some operating systems (such as Linux and FreeBSD) it is possible to use extended attributes to create a *searchable* attribute in the ACL, which, if used for partitioning, would offer the same benefits as Security Aware Partitioning. Thus we leave the application of our partitioning algorithm to standard ACLs for future work and focus on the traditional UNIX permissions in this paper.

2.1.2. Using Permissions for Partitioning. For search to be secure, a file system needs to be aware of a combination of *read*, *write*, and *execute* permissions for all security levels. For a user on a UNIX system to access files or subdirectories within a directory, the directory's *execute* bit must be set for some role which the user fulfills. If the *other* execute bit is set, any user who is not the owner or a member of the group can access files in or below that directory. Otherwise the user must be the file's owner or a member of the group, and have the corresponding *execute* bit set. Further, the *execute* bit must be set on every directory preceding the current one in the path. In other words, access is determined by the logical AND of the access permissions of every directory in a file's path, relative to a specific user's roles.

However, for a user to actually view the contents of a directory, they must not only have *execute* permission, but *read* permission as well. Unlike *execute*, *read* does not require permissions all along the path. It is possible to read the contents of a directory as long as the user has *read* permissions on the last directory in the path. This allows security operations such as permitting users to own a directory without being able to list the contents of directories above that one. For instance, a system may choose to not let users view which other users have directories in `/home`, even though the users themselves are the owners of `/home/<username>` directories. This is similar to the model for file search security developed by Büttcher [12].

Therefore, Security Aware Partitioning partitions the file system according to *group* and *user* security permissions. The algorithm walks the file system in a breadth first search. Access permission is determined by examining all permissions in the directories above the file or directory in question. If the permissions on the current file or directory are more restrictive than that of the current partition or the *user* or *group* has changed, then a new partition is created. Subfolders and files are added until another restriction in permissions is encountered. For example, if the current directory has permissions 700, but the parent directory has permissions 777, then only a subset of users who could access the parent directory will be able to access the current directory, and a new partition must be created. This ensures that all files and directories in a given partition can be accessed by the same set of users.

2.2. Other Partitioning Algorithms

In order to evaluate the effectiveness of Security Aware Partitioning, we compared it to a wide variety of other possible partitioning algorithms. Three of the algorithms we evaluated are based on likely user preference: *greedy time*, *interval time*, and *user based*. Since prior research, such as Stuff I've Seen [16], suggests that users prefer recent files over older ones, partitioning on time would align well with user interests.

We implemented two different time based algorithms, one which we call *greedy time* and the other *interval time*. The greedy algorithm creates partitions by starting at the most recently modified file and moves backwards in time, filling partitions of 100,000 files each. It is important to note that this count does not include directories; the size of the partition is determined by files only. This is an empirical number that Leung et al. [24] suggests creates indexes appropriately sized for main memory. The interval based algorithm partitions files for the past day, the past week, the past month, and so forth, excluding files already consumed. The intervals we chose are shown in Table 1. These intervals reflect the quarter system that companies and UCSC use; the future interval exists to handle cases where some files have modification times in the future.

Table 1. Time Intervals for Interval Time Algorithm

Time Intervals
future
past day
past week
past month
past 3 months
past 6 months
past 9 months
past year
past 2 years
past 5 years
past 10 years
older than 10 years

Similarly, *user based partitioning* takes advantage of the fact that users are likely to prefer their own files. Our implementation gives each user his/her own partition, adding files to partitions based on the owner of the file.

Cosine correlation clustering measures the similarity between two vectors (in this case, the metadata for two files or clusters of files) by finding the cosine of the angle between them, using the equation

$$similarity = \cos(\theta) = \frac{A \cdot B}{||A||\,||B||} \tag{1}$$

where A and B are vectors of metadata. If the correlation of the metadata is above some threshold constant ϵ, implying a strong similarity in metadata, the two vectors are merged into a new cluster, and the centroid vector of the cluster is added back into the pool for comparison. Clustering can be done by finding a best match or simply accepting the first match discovered. We opted for the latter due to the lower computational complexity. We chose this algorithm because we wanted to replicate the results of *SmartStore* [21], which attempts to improve performance by clustering data according to the correlation of its metadata, and then using the clusters as the partitions. It does this by combining cosine correlation clustering with *Latent Semantic Analysis (LSA)* [15].

Latent Semantic Analysis is a technique for grouping related attributes, commonly used in information retrieval to discover correlation, or conceptual relations between terms. When used in information retrieval, LSA creates a matrix of the number of times unique terms occur in each document. For terms and documents, this usually results in a very sparse matrix, since most words are in few files. *Singular Value Decomposition (SVD)* [17] is applied to this matrix resulting in three separate matrices: a term-concept matrix, a singular values matrix, and a transformed concept-document matrix. LSA reduces the number of concepts in the concept-document matrix to a value of k, which eliminates the noise of the document while preserving the semantic information of the documents.

SmartStore, instead, creates a matrix of attributes and files. All files will have all the POSIX attributes, so the matrix is denser than a terms/documents matrix. SmartStore first applies LSA to a subset of the metadata to create a transformation, which is then applied to the rest of the data. Partitions are created by examining the cosine correlation of the transformed data and grouping files which have a correlation greater than the threshold constant ϵ. Since LSA is a relatively computationally intensive algorithm, we evaluated cosine clustering with and without LSA in order to determine what advantages LSA brings.

For both of the correlation based algorithms, a suitable choice of ϵ was needed. We did statistical analysis with a subset of the data (between 0.5 and 1%) to determine the mean and standard deviation of the correlation. We then selected an ϵ the square of one standard deviation above the mean, which empirically generated an appropriate set of partition sizes.

Spyglass [24], a metadata search system, uses a greedy depth first search when creating partitions. Files and directories are added to the partition with the longest pathname match. New partitions are created when all matching partitions are "full", defined as containing over 100,000 files. The authors show Spyglass having an improvement in search performance of between one and four orders of magnitude in comparison with database management systems. To test this claim against other partitioning algorithms designed for file system search, we implemented a greedy depth first search algorithm.

3. Experimental Design

To compare the various partitioning algorithms described in Section 2, we ran each algorithm over four different anonymized file system metadata crawls, summarized in Table 2. One, *SOE*, was collected from the School of Engineering at the University of California, Santa Cruz; the other three, *Web*, *Eng*, and *Home*, were collected from various file servers at NetApp, Inc. The crawls contain 11 metadata attributes, enumerated in Table 3. The crawls do not contain any content information.

Table 2. Crawl Descriptions

Crawl	Description	# of Files
SOE	file server	6901466
NetApp Web	web/wiki	15569242
NetApp Eng	engineering scratch	60432243
NetApp Home	home directories	268539360

We evaluated the resulting partitions using the following criteria: size of the partitions, runtime and memory usage (evaluated using Big-O run times, in order to account for variations in the algorithms), the actual files within the partitions, partition entropy, and information gain. By looking at the size of the partitions, the runtime and memory usage, and the files within the partitions, we can gain an understanding of how the file system would be partitioned. Partition entropy and information gain can be used to understand how the partitioned file system might perform if the system was queried for a given attribute. These criteria were selected because it allows the partitioning algorithms to be compared without building complete systems with working implementations of each algorithm.

Since there are currently no standard benchmarks for file system search, many systems have adopted the method of generating randomized queries in order to evaluate performance [24], [21]. Unfortunately, most of these queries are not representative of what real users will ask. Information retrieval teaches us that users focus on certain attributes and specific values [13]. Without conducting a study of user querying behavior, it is impossible to know which attributes and values will be popular for a given system and data set. We therefore chose the criteria in an effort to fully characterize what a system with a specific algorithm implemented would, and would not, be good at. We do not intend for these criteria to replace performance benchmarking, but rather serve as a complement–a way of identifying which algorithms to investigate further through implementation in a fully working system.

3.1. Criteria

In the rest of the paper, we define entropy to be the variance of attribute values in a given partition. In other words, entropy measures the number of values of an attribute within a partition. An entropy of zero means that all instances of that attribute have the same value in that partition. To calculate entropy, we used the Shannon formula for information [30]:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i) \tag{2}$$

We define information gain as the difference between the entropy of the whole data set and the entropy of individual partitions, calculated by attribute. Information gain, then, is the amount of information you gain about an attribute by being in a given partition. A high information gain indicates that the values for that attribute are generally unique to that partition. We used information gain as described by Quinlan [28]:

$$H(X) - H(X|p) \text{ for each partition p} \tag{3}$$

For each attribute, we calculated the average information gain over all partitions. This allows us to look at the whole system; a high average information gain on a given attribute implies that values of that attribute are generally concentrated in unique partitions.

Finally, we compared the files in the partitions generated by each algorithm to the partitions generated by each other algorithm. We use an intersection metric that measures how many files from a partition under one algorithm are contained in the best match partition from another. This captures both differences in content and size of partitions. Note that this measure is not symmetric, since the best match may not be the same in both directions, and therefore we present intersection results both ways for a given pair of algorithms.

Table 3. Attribute Descriptions

Name	Description
path	path to file
type	file extension
inode	inode number
mode	security permissions
links	number of hard links
uid	user id
gid	group id
size	file size
atime	access time
mtime	modification time
ctime	creation time

4. Results and Analysis

In the interests of brevity, we only present results for the SOE and NetApp Web/Wiki data sets. We discuss partition sizes, compare the contents of partitions across partitioning algorithms, and analyze the information density. We explore the implications for each of these results. We show that Security Aware Partitioning has minimal space and time requirements, as well as good query performance, as judged by entropy and information gain, regardless of the metadata used in the query.

In addition, we determine that partition sizes have high variance for every algorithm except those with fixed sizes, leading to poorer index performance. Finally, we show that LSA and cosine correlation clustering are nearly indistinguishable, given the right choice of constants.

4.1. Runtime

The runtime in Table 4 is presented in Big-O notation, in order to compensate for factors which can cause the runtime to vary. In general, lower runtimes are preferred, since they reduce the overhead for the system to support search.

Security Aware Partitioning is linear with respect to the size of the data set. Files and directories do not need to be compared to other files/directories, just to the security permissions of the current partition. Because UNIX style permissions rely on the hierarchy, we do a recursive tree descent. The permissions of each file in the path are stored on the stack, for a memory requirement no greater than the depth of the file system, $log_b(n)$, where n is the number of files, and the base of the log, b, is the branching factor. The wide branching factor of directories makes the constants quite low in practice. This memory usage is a constraint specific to UNIX style permissions, since they rely on parent directories, and would not necessarily apply to other architectures.

Greedy algorithms, by their nature, require very little time and space. The current file is compared to the count of files in the existing partition, and either the partition is full and a new partition is created, or the file is added to the current partition. The memory needed for user partitioning is predicated on the number of users in the system being fairly constant and significantly smaller than the number of files.

Both cosine correlation and LSA with cosine correlation require that every file in the data set potentially be compared to every other file. In addition, LSA requires a singular value decomposition operation (SVD), which runs in $O(npq)$, where p is the number of metadata elements (one for each numeric attribute, and one for each possible value of a non-numeric attribute), and q is the number of dimensions to reduce to. While this is not expensive for basic numeric UNIX metadata, it quickly becomes prohibitively expensive for extended metadata and non-numerical metadata such as user. These attributes need to be treated as a series of binary attributes, one for each possible value, in order to correctly perform SVD.

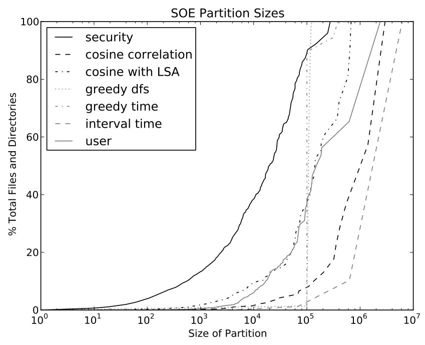
4.2. Partition Size

Partition sizes directly impact the effectiveness of search. If the partitioning algorithm results in a large number of small partitions, then finding relevant files may result in many calls to the disk in order to load all of the required indexes. By contrast, if partitions are too large then the index cannot easily fit into memory. An ideal partitioning algorithm will result in most partitions being near the maximum size, with a fairly low variance in size. We show the mean, median, and variance for partition size in Table 5 and Table 6.

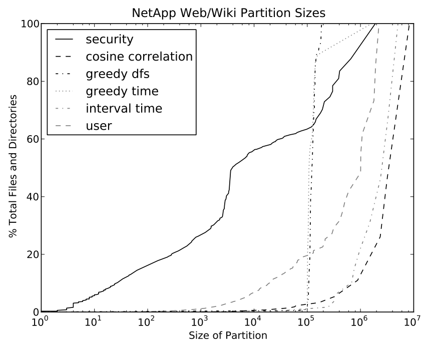
As mentioned in Section 2.2, the greedy algorithms used a fixed number of files to determine the partition size, not including directories. This is why there is a standard deviation associated with the greedy algorithms, and why not every partition had a size of exactly 100,000. As expected, the non-greedy algorithms had a large number of smaller partitions and a few large partitions, as shown in Figures 2(a) and 2(b). The skewed nature of metadata has been explored by Leung [24], and any algorithm which relies on it is likely to be somewhat skewed in distribution. This means any non-greedy algorithm will construct many smaller indexes that will later need to be accessed. However, this cost can potentially be mitigated through clever on-disk layout and data structures. Non-greedy algorithms can also result in partitions which are over

Table 4. Runtime analysis. n is the number of files

	Greedy DFS	Greedy Time	Interval	User	Security	Cosine	LSA
Time	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$O(n^2) - O(n^3)$
Memory	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$	$O(n)$	$O(n)$



(a)



(b)

Figure 2. CDFs for the partition sizes of different partitioning schemes. (a) SOE – Security Aware Partitioning produces many small partitions, but very few that are over the 100,000 mark. (b) NetApp Web/Wiki – Over half of the partitions created by Security Aware Partitioning are smaller than 100,000.

100,000 files and may be too large for indexing. In this case, a secondary algorithm (such as partitioning by modification time), could be used to split the partition into more manageable sub-partitions. Further investigation on the use of secondary algorithms remains as future work; in this paper we focus on identifying a good primary algorithm.

LSA and cosine correlation are similar for some data sets. For the Web/Wiki data, the mean and standard deviation for partition sizes are identical. (Recall that the size of partitions is governed by the choice of constant, ϵ.) For the SOE data, they are more dissimilar, suggesting that the algorithm may have found more correlation to exploit.

Security Aware Partitioning has a low standard deviation, suggesting that partitions tend to be approximately the same size. However, the mean size is at least an order of magnitude lower than any of the other algorithms. This means Security Aware Partitioning creates a large number of small partitions. A possible solution to this would be to merge partitions that have the same set of users who can access them, eliminating the hierarchical boundaries that are currently in place. This requires a more advanced version of the Security Aware Partitioning algorithm, and is part of our future work.

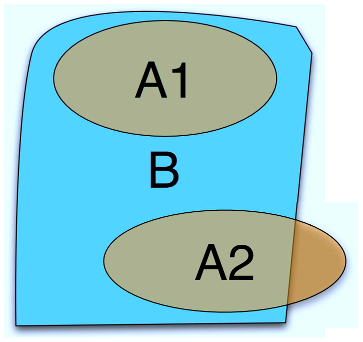


Figure 3. Comparing content. A1 and A2 are the same size, while B is much larger. A1 is fully contained in B, but is only 30% of B, so the pairwise comparison A1/B would be 0%/70%. A2 is not fully contained in B—20% is different—so the pairwise comparison A2/B would be 20%/75%. If, however, A1 and A2 were merged into a single partition, then the pairwise comparison A/B becomes 10%/45%. This can be used to infer that partitions generated by algorithm A are very similar to those generated by B, since they divide up the data in a similar fashion.

4.3. Partition Content

If two algorithms place the same files in the same partitions, all other things being equal, they will have similar costs for a given query. Therefore, compar-

Table 5. SOE Size Statistics. Greedy algorithms did not count directories in the size determination, thus the mean size is not exactly 100,000 and there is a standard deviation.

	Greedy DFS	Greedy Time	Interval	User	Security	Cosine	LSA
Number of partitions	81	64	8	384	29479	131	1370
Mean size	85203	107835	862683	17973	234	5037	52683
Standard Deviation	44487	43634	2163134	128252	3309	36776	292193

Table 6. NetApp Web/Wiki Size Statistics. Greedy algorithms did not count directories in the size determination, thus the mean size is not exactly 100,000 and there is a standard deviation.

	Greedy DFS	Greedy Time	Interval	User	Security	Cosine	LSA
Number of partitions	156	125	12	1908	318782	140	140
Mean size	99802	124553	1297437	8159	48	111208	111208
Standard Deviation	2463	134993	1552147	92735	3930	777795	777795

ing the content of partitions is a useful metric for comparing the behavior of partitioning algorithms. In order to evaluate content similarities, we opted for an intersection metric, since it would capture variations in both content and size of partitions. Since we used a intersection metric, results are not symmetric and should be considered in pairs: how well X is contained by Y versus how well Y is contained by X. Figure 3 shows an example of how to interpret the data. In Tables 7 and 8, two low numbers in the same cell indicate the partitioning algorithms generate partitions similar in content and size, while two high numbers in the same cell indicate the algorithms generate partitions different in both content and size. A low number and a high number indicates similar content, but different size partitions.

Note in Table 7 that cosine correlation compared to LSA is very similar, with a difference of 9.9%. Conversely, LSA compared to cosine correlation has a difference of 60.1%. This suggests that for every one partition created by cosine correlation clustering, the LSA algorithm puts the same information in multiple partitions. This seems reasonable, given the disparity in partition sizes between LSA and cosine correlation. This means that the two will access a similar proportion of indexes for a given query, but LSA will have to load more indexes in total.

By contrast, the greedy time algorithm and the greedy DFS algorithm have a very symmetric difference in Table 7, around 66% in both directions. Since they have very similar partition sizes, this suggests that the contents of partitions are very different for these two algorithms, and will have very different index accesses for a query. The comparison numbers for Security Aware Partitioning are around 10% for almost all the other algorithms (excluding greedy time), indicating that the partitions generated are similar in content but not in size. This result makes sense, since Security Aware Partitioning generates a large number of smaller partitions (we discuss mitigation strategies in future work). Based on this, we can conclude that Security Aware Partitioning will access similar proportions of indexes to other algorithms.

4.4. Partition Entropy and Information Gain

Partition entropy and information gain help estimate the effectiveness of the partitioning method for search. Partition entropy measures the "goodness" of a partition, by measuring the entropy per attribute within each partition. This measures the number of values of an attribute in a given partition. A low entropy suggests that the attribute values within that partition are somewhat homogeneous – there are only a few attribute values in that partition.

Information gain is the difference between the entropy of the whole data set and the entropy of individual partitions and is calculated on a per attribute basis. High information gain indicates the attribute values found within that partition are highly concentrated in that partition, meaning that most of the files with a specific attribute value can be found there.

For entropy calculations, we did not include the path name or the inode number, since these will almost always be unique to a specific file or directory. In Figures 4 and 5 we present the cumulative distribution function of entropy for different attributes, with each algorithm displayed. Here, a fast growth rate implies that most of the entropy for that algorithm was low, and therefore the algorithm will be more efficient at retrieving data related to that attribute. We have selected a few attributes from the SOE data to display, based on common user queries.

Table 7. SOE Partition Content Comparison. Each entry for row X, column Y, can be read as "% average of X in Y/% average of Y in X". Items of particular interest have been highlighted. Security is not significantly different from most other algorithms, about 10% on average, but is significantly different from greedy time. Cosine and LSA are very similar to one another.

	Greedy DFS	Greedy Time	Interval	User	Security	Cosine	LSA
Greedy DFS	0%	66.9/65.9%	10.3/97.3%	28.5/56.4%	60.3/10.6%	7.4/79.8%	26.6/42.6%
Greedy Time	65.9/66.9%	0%	1.2/92.0%	56.5/74.3%	78.0/41.5%	40.0/85.3%	60.9/68.9%
Interval	97.3/10.3%	92.0/1.2%	0%	65.3/10.3%	95.7/10.0%	56.7/10.3%	89.3/10.3%
User	56.4/28.5%	74.3/56.5%	10.3/65.3%	0%	71.1/14.2%	24.2/52.8%	49.0/33.7%
Security	10.6/60.3%	41.5/78.0%	10.0/95.7%	14.2/71.1%	0%	4.6/83.1%	13.3/63.6%
Cosine	79.8/7.4%	85.3/40.0%	10.3/56.7%	52.8/24.2%	83.1/4.6%	0%	60.1/9.9%
LSA	42.6/26.6%	68.9/60.9%	10.3/89.3%	33.7/49.0%	63.6/13.3%	9.9/60.1%	0%

Table 8. NetApp Web/Wiki Partition Content Comparison. Each entry for row X, column Y, can be read as "% average of X in Y/% average of Y in X". Items of particular interest have been highlighted. For this data set, LSA partitions are identical to cosine correlation, and therefore LSA makes no difference. Security is more distinct from other schemes for this data set, but extremely similar to the more expensive LSA.

	Greedy DFS	Greedy Time	Interval	User	Security	Cosine	LSA
Greedy DFS	0%	79.3/82.1%	46.1/94.3%	31.3/79.0%	59.5/27.4%	2.5/92.4%	2.5/92.4%
Greedy Time	82.1/79.3%	0%	1.9/82.1%	56.2/80.4%	74.5/55.4%	40.5/88.9%	40.5/88.9%
Interval	94.3/46.1%	82.1/1.9%	0%	72.4/49.0%	83.9/28.3%	46.4/66.5%	46.4/66.5%
User	79.0/31.3%	80.4/56.2%	49.0/72.4%	0%	65.0/12.3%	16.9/67.9%	16.9/67.9%
Security	27.4/59.5%	55.4/74.5%	28.3/83.9%	12.3/65.0%	0%	1.5/81.6%	1.5/81.6%
Cosine	92.4/2.5%	88.9/40.5%	66.5/46.4%	67.9/16.9%	81.6/1.5%	0%	0%
LSA	92.4/2.5%	88.9/40.5%	66.5/46.4%	67.9/16.9%	81.6/1.5%	0%	0%

Table 9. SOE Average Information Gain in bits

Algorithm	type	mode	links	uid	gid	size	atime	mtime	ctime
Greedy DFS	3.5	1.5	0.3	2.4	1.8	6.6	3.2	7.0	9.0
Greedy Time	7.1	3.0	0.5	4.8	3.7	13.2	6.4	14.1	18.2
Interval	6.3	2.7	0.5	4.2	3.3	11.7	5.7	12.5	16.1
User	3.6	1.5	0.3	2.4	1.8	6.7	3.2	7.1	9.1
Security	7.2	3.0	0.5	4.8	3.7	13.4	6.5	14.3	14.3
Cosine	7.1	3.0	0.5	4.8	3.7	13.2	6.4	14.1	18.1
LSA	7.2	3.0	0.5	4.8	3.7	13.4	6.5	14.2	18.4

The information gain is presented in Tables 9 and 10 for each attribute, so that the quality of partitioning can be evaluated for different types of searches. High information gain indicates that partitions mostly contain a single or small number of attribute values. Algorithms which partition over a specific attribute are likely to have good information gain for that attribute. For instance, the greedy time algorithm has excellent information gain for modification time (mtime) since it partitions based on that attribute. However, a good partitioning criteria will also have high information gain for other attributes. Cosine correlation's information gain is slightly lower than cosine correlation with LSA, suggesting that LSA is slightly better, but may not garner sufficient additional benefits to justify the added computation. Security Aware Partitioning has good information gain for all attributes, and consistently outperforms all other algorithms.

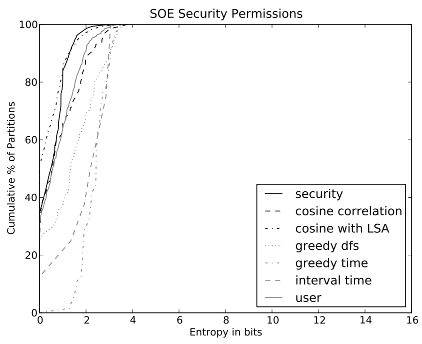
5. Related Work

In addition to the algorithms we have compared in this paper, there has been a great deal of prior research into partitioning indexes, both for file systems and web search. We mention here other work in addition to the algorithms we evaluated.

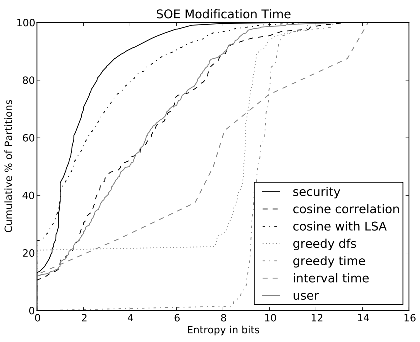
Security for search is a complex area. We have focused particularly on desktop and enterprise search

Table 10. NetApp Web/Wiki Server Average Information Gain in bits.

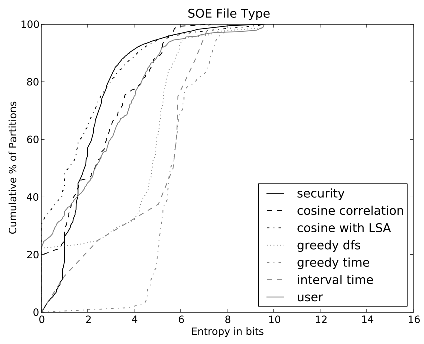
Algorithm	type	mode	links	uid	gid	size	atime	mtime	ctime
Greedy DFS	3.2	2.6	1.0	5.1	0.6	4.2	0.0	12.8	8.9
Greedy Time	3.2	2.6	1.0	5.1	0.6	4.2	0.0	12.8	8.9
Interval	2.7	2.2	0.9	4.3	0.5	3.5	0.0	10.9	7.6
User	3.2	2.6	1.0	5.1	0.6	4.2	0.0	12.8	8.9
Security	3.2	2.6	1.0	5.1	0.6	4.2	0.0	12.8	8.9
Cosine	3.2	2.6	1.0	5.0	0.6	4.2	0.0	12.6	8.8
LSA	3.2	2.6	1.0	5.0	0.6	4.2	0.0	12.6	8.8



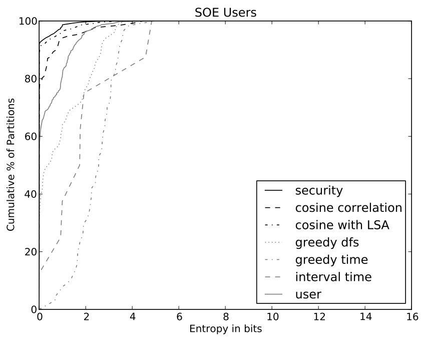
(a)



(b)



(c)



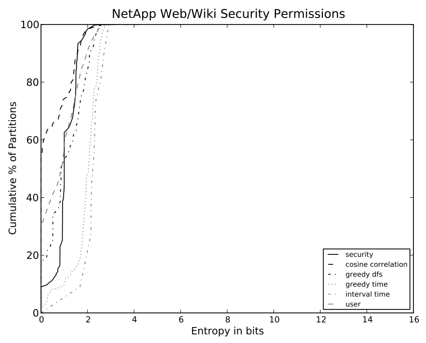
(d)

Figure 4. SOE entropy in bits. CDFs of entropy by (a) mode, (b) mtime, (c) type, and (d) uid for percentage of partitions. Algorithms which grow more quickly in this graph are better for search. Note that the security algorithm grows quickly, meaning it has excellent entropy for all attributes.

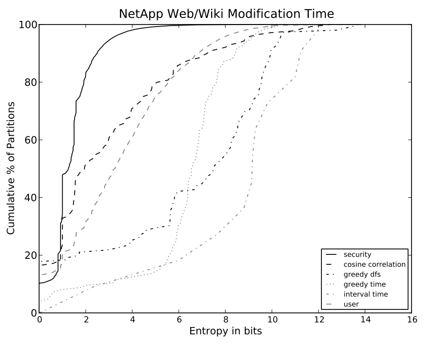
in our review of related work. However, large file system search combines aspects of both of these and is an under-explored area of research.

5.1. Partitioning

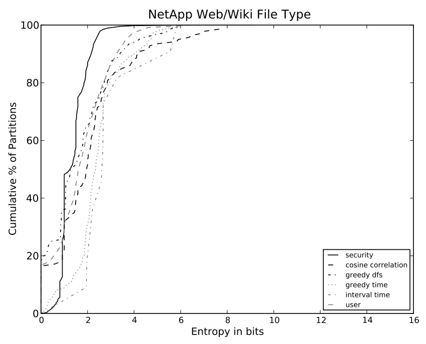
One of the first systems to propose a search technique similar to partitioning was GLIMPSE [25]. It was designed for full text search over a file system, and was created to reduce the cost of brute force search without incurring the space costs of a full text index. GLIMPSE used a dictionary over large areas of a file system. Once an area of the file system was identified that contained the search term, a brute force search would be carried out within the area to find the individual documents that satisfied the query. This is similar to current techniques for file system partitioning. However, it still required a brute force search once the correct partition was identified, making it necessary to access the disk for the contents of a large number of files. By contrast, our system only requires that the metadata index be loaded into memory



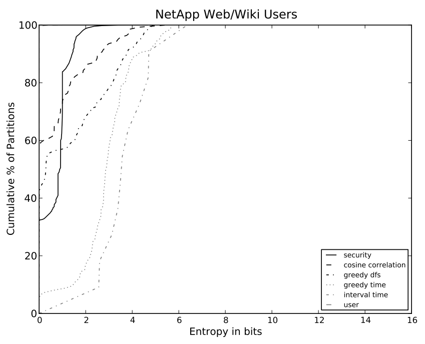
(a)



(b)



(c)



(d)

Figure 5. Web/Wiki entropy in bits. CDFs of entropy by (a) mode, (b) mtime, (c) type, and (d) uid for percentage of partitions. Interestingly, security performs less well for permissions on this data set, but still does well over-all.

(or already be resident in memory), for significantly less overhead.

Lester et al. proposed geometric partitioning [23] in order to allow full text search indexes to be updated online. A series of increasingly large indexes would store documents, from newest to oldest, favoring newer documents. However, all partitions needed to be searched in order to return results; partitioning simply amortized the cost of merging indexes. Rather than using variably sized hierarchical indexes, we use parallel indexes and only require partitions to be searched if they contain the metadata the user is looking for.

One of the first examples of user partitioning is found in the *Rapid Access Storage Hierarchy* (RASH) file system [19]. RASH grouped files from each user on volume sets separate from other users when archiving, so each user had dedicated volume sets of his/her files. Our implementation takes the same approach and gives each user his/her own partition, adding files based on the owner of the file.

Carmel et al. [14] proposed a lossy method for pruning indexes, term-based pruning, where term listings below some threshold would be eliminated. This was designed to reduce index size for small environments such as PDAs.

Other systems have focused on index pruning as a form of multi-level caching – maintaining a full index but only keeping some fraction of it in top level storage and maintaining the rest in slower storage. For instance, document centric pruning [13], proposed by Büttcher and Clarke, was a development on term-based pruning. Rather than retaining the top k entries in a posting list, they retained the top k term postings for a given document. Query based partitioning [26] takes this idea one step further, calculating which queries are likely, based on prior query history, and retaining only the top k documents for each likely query.

5.2. Security

Conventional desktop search strategies are not designed to deal with such a large volume of data, and security is generally not a concern. Most desktop search systems do handle security but at the price of constructing indexes for every user, such as Copernic [1], MSN Toolbar [2], Yahoo Desktop [3], and Google Desktop [4]. This strategy is not scalable to the enterprise level. There are a large number of files on an enterprise system that everyone can access, and there are other smaller subsets which are shared among groups and single users. A per user index results in an untenable amount of duplicate data from shared files.

Büttcher et al. have explored the question of security models for file system search [12]. However, their proposed system, Wumpus, requires a decrease in search performance in order to achieve security and requires a high overhead security manager. By contrast, we propose to both offer secure search and improve search performance.

This problem is also similar to the problem of enterprise search, where the user is searching over a variety of documents with differing security permissions in a corporate intranet. Some research has been done on this subject, notably Bailey, Hawking and Matson's paper on document level security [9]. Bailey et al. propose access checking at the "collection" level for intranet documents, but at the cost of accurate match counts.

According to Bailey et al. [9], Enterprise search technologies such as Google Search Appliance, Coveo, and FAST ESP (formerly Convera RetrievalWare) have some awareness of security, but take performance hits, consume more space, and/or suffer accuracy penalties to offer it. Security must be painstakingly and manually applied or extracted from external security systems, a task which can lead to high latency, and slowdowns for document filtering and re-ranking. According to Google, Google Search Appliance, for instance, performs late-binding security in which access control checks are performed against the content hosts in real time when the query is executed [18]. And, as with other web search, even the best enterprise search systems do not take advantage of the rich metadata available on the file system.

6. Non-hierarchical File Systems and Their Implications

There is a clarion call for file systems that no longer rely on hierarchy to organize content. From Inversion [27] and LiFS [7] to iTunes [8] and Seltzer's position paper that "hierarchical file systems are dead" [29], systems are moving towards rich metadata and search for organizing and finding content. On such a file system, fast high quality search over metadata and content is more important than ever, since the hierarchy can no longer be used to navigate and find information. How well do these algorithms perform when the file system no longer has explicit hierarchy?

Any algorithm that relies explicitly on hierarchy is clearly out of the question, such as a greedy depth first search method. However, methods which rely on properties of the data itself, such as the owner, the security permissions, or the time, will continue to create similar partitions and be just as effective.

While UNIX file system permissions currently rely on the hierarchy to describe access rights, there is no reason to believe that users will change the way they assign access rights, simply because the organization of the file system has changed. Users and groups will continue to guide who can see what files, and therefore Security Aware Partitioning is a strategy with long term potential, no matter the underlying file system organization.

7. Future Work

We will be extending this work in the future to explore content and richer metadata, especially semantic and security metadata. We intend to implement a fully functional prototype of this system using Ceph [31] as a foundation. While metadata search is important, full text and keyword search is an integral part of any search system. We will explore the effectiveness of our algorithm applied to full text search in addition to metadata. We also intend to explore the impact of rich metadata on the performance of partitioning systems. We would like to explore whether other metadata can provide partitions as optimal as Security Aware Partitioning.

Access Control Lists (ACLs) can be used to implement a much richer security model, and we would like to explore real world usage of ACLs, to determine the implications for security based partitioning.

8. Conclusions

As the volume of file systems increases, the performance of search becomes increasingly important. Algorithms which are good on a thousand files may be painfully slow for a hundred million. Partitioning algorithms are an effective way to reduce the cost of search, but these algorithms must also scale well, or the cost of building indexes will overwhelm the system.

The choice of a partitioning algorithm will always depend on the requirements of a specific system. For some systems, security may be a non-issue and temporal access the dominant criteria. In these systems, a greedy algorithm is likely the best solution, due to its creation of fixed size partitions. For other systems, security may be mission critical. However, for any system, performance is always a first concern. We have proposed a series of metrics which can be be used to evaluate the expected performance of partitioning algorithms without actually building the search indexes and testing with generated queries, which may not be representative of real user needs. In addition, we have measured the performance of a variety of current algorithms using these metrics. We anticipate these will be useful to others seeking to design and benchmark future algorithms.

Not only must file system search be scalable and high performance, it must respect the security constraints of the file system over which it operates. Users should never be able to see files they cannot access. By combining these two necessary qualities of search, we have developed a new partitioning algorithm that will scale as file systems grow. Security Aware Partitioning respects the file system's security settings, creating partitions in which all files are accessible by the same set of people. By early elimination of areas users cannot see, we can speed up search significantly. Due to the simplicity of our algorithm, we also reduce overhead at indexing time. Furthermore, the partitions generated by Security Aware Partitioning have good properties for search. They have low entropy, high information gain, and are similar to partitions created by other, more computationally expensive algorithms. We have shown that partitioning with respect to security permissions is both secure and efficient, making Security Aware Partitioning a strong partitioning algorithm for file system search.

Acknowledgements

This research was supported in part by the National Science Foundation under awards CCF-0937938 and IIP-0934401 (I/UCRC Center for Research in Intelligent Storage), by the Department of Energy's Petascale Data Storage Institute under award DE-FC02-06ER25768, and by the Center for Information Technology in the Interest of Society (CITRIS). We also thank the industrial sponsors of the SSRC, including NetApp, IBM, Symantec, and LSI for their generous support, and NetApp in particular for generously making data sets available to us.

References

[1] Copernic. http://www.copernic.com.

[2] MSN toolbar. http://toolbar.msn.com/.

[3] Yahoo desktop. http://desktop.yahoo.com/.

[4] Google desktop. http://desktop.google.com, January 2009.

[5] U.S. Department of Health & Human Services: Health Information Privacy. http://www.hhs.gov/ocr/privacy/, Jan 2010.

[6] U.S. Securities and Exchange Commission. http://www.sec.gov/, Jan 2010.

[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] BAILEY, P., HAWKING, D., AND MATSON, B. Secure search in enterprise webs: Tradeoffs in efficient implementation for document level security. In *Proceedings of the 2006 International Conference on Information and Knowledge Management Systems (CIKM '06)* (Nov. 2006).

[10] BLOOM, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13*, 7 (July 1970), 422–426.

[11] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual web search engine. *Computer Network and ISDN Systems 30*, 1-7 (1998), 107–117.

[12] BÜTTCHER, S., AND CLARKE, C. L. A. A security model for full-text file system search in multi-user environments. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST)* (San Francisco, CA, December 2005).

[13] BÜTTCHER, S., AND CLARKE, C. L. A. A document-centric approach to static index pruning in text retrieval systems. In *Proceedings of the 2006 International Conference on Information and Knowledge Management Systems (CIKM '06)* (2006), pp. 182–189.

[14] CARMEL, D., COHEN, D., FAGIN, R., FARCHI, E., HERSCOVICI, M., MAAREK, Y. S., AND SOFFER, A. Static index pruning for information retrieval systems. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '01)* (2001), pp. 43–50.

[15] DEERWESTER, S., DUMAIS, S. T., FURNAS, G. W., LANDAUER, T. K., AND HARSHMAN, R. Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE 41*, 6 (1990), 391–407.

[16] DUMAIS, S., CUTRELL, E., CADIZ, J., JANCKE, G., SARIN, R., AND ROBBINS, D. C. Stuff I've seen: a system for personal information retrieval and re-use. In *Proceedings of the 26th annual international ACM SIGIR conference on Research*

and development in informaion retrieval (2003), ACM Press, pp. 72–79.

[17] GOLUB, G. H., AND VAN LOAN, C. F. *Matrix Computations*. Johns Hopkins University Press, 1996.

[18] GOOGLE. Managing search for controlled-access content: Overview. http://code.google.com/apis/searchappliance/documentation/62/secure_search/secure_search_overview.html, October 2009.

[19] HENDERSON, R. L., AND POSTON, A. MSS-II and RASH: A mainframe UNIX based mass storage system with a rapid access storage hierarchy file management system. In *Proceedings of the Winter 1989 USENIX Technical Conference* (1989), pp. 65–84.

[20] HUA, Y., JIANG, H., ZHU, Y., FENG, D., AND TIAN, L. SmartStore: A new metadata organization paradigm with metadata semantic-awareness for next-generation file systems. Tech. Rep. UNL-CSE 2008-0012, University of Nebraska, Nov. 2008.

[21] HUA, Y., JIANG, H., ZHU, Y., FENG, D., AND TIAN, L. SmartStore: A new metadata organization paradigm with semantic-awareness for next-generation file systems. In *Proceedings of SC09* (Nov. 2009).

[22] KLEINBERG, J. M. Authoritative sources in a hyperlinked environment. *J. ACM 46*, 5 (1999), 604–632.

[23] LESTER, N., MOFFAT, A., AND ZOBEL, J. Fast on-line index construction by geometric partitioning. In *Proceedings of the 2005 International Conference on Information and Knowledge Management Systems (CIKM '05)* (Nov. 2005).

[24] LEUNG, A., SHAO, M., BISSON, T., PASUPATHY, S., AND MILLER, E. L. Spyglass: Fast, scalable metadata search for large-scale storage systems. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)* (Feb. 2009), pp. 153–166.

[25] MANBER, U., AND WU, S. Glimpse: A tool to search through entire file systems. Tech. Rep. TR 93-94, The University of Arizona, Oct 1993.

[26] MITRA, S., WINSLETT, M., AND HSU, W. W. Query-based partitioning of documents and indexes for information lifecycle management. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (June 2008).

[27] 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.

[28] QUINLAN, J. R. Induction of decision trees. *Mach. Learn. 1*, 1 (1986), 81–106.

[29] SELTZER, M., AND MURPHY, N. Hierarchical file systems are dead. In *Proceedings of the 12th Workshop on Hot Topics in Operating Systems (HotOS-XII)* (2009).

[30] SHANNON, C. E., AND WEAVER, W. *Mathematical Theory of Communication*. University of Illinois Press, 1963.

[31] 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)* (Seattle, WA, Nov. 2006), USENIX.