Wayfinder: Navigating and Sharing Information in a Decentralized World

Christopher Peery, Francisco Matias Cuenca-Acuna,
Richard P. Martin, Thu D. Nguyen

Department of Computer Science, Rutgers University, Piscataway, NJ 08854
{peery, mcuenca, rmartin, tdnguyen}@cs.rutgers.edu

Appears in Proceedings of DBISP2P 2004

Abstract. Social networks offering unprecedented content sharing are rapidly developing over the Internet. Unfortunately, it is often difficult to both locate and manage content in these networks, particularly when they are implemented on current peer-to-peer technologies. In this paper, we describe Wayfinder, a peer-to-peer file system that targets the needs of medium-sized content sharing communities. Wayfinder seeks to advance the state-of-the-art by providing three synergistic abstractions: a global namespace that is uniformly accessible across connected and disconnected operation, content-based queries that can be persistently embedded into the global namespace, and automatic availability management. Interestingly, Wayfinder achieves much of its functionality through the use of a peer-to-peer indexed data storage system called PlanetP: essentially, Wayfinder constructs the global namespace, locates specific files, and performs content searches by posing appropriate queries to PlanetP. We describe this query-based design and present preliminary performance measurements of a prototype implementation.

1 Introduction

Social networks offering unprecedented content sharing such as Gnutella, KaZaA, and DMOZ are rapidly developing over the Internet. Unfortunately, locating specific information in these networks can often be frustrating, particularly when they are implemented using current peer-to-peer (P2P) technologies. For example, the Direct Connect system (http://www.neo-modus.com) used by a local file sharing community allows the browsing of each individual node's shared content but does not support a global browsable namespace. Consequently, over 25TB of data is completely fragmented across more than 8000 individual listings without any way for the users to collaboratively organized this shared information. To exacerbate the problem, content search is quite primitive, content ranking is not supported, and it is impossible to reason about data availability because of extensive disconnected operation[1]. This state-of-the-art is common to other P2P systems such as KaZaA (http://www.kazaa.com) and eMule (http://www.emule-project.net).

Managing shared content is also difficult, particularly when users participate in multiple networks, as users must manually track content replicas across multiple publishing

[1] Consistent with [2, 24], traces show that most users connect for less than a few hours a day.

infrastructures and their local writable storage systems. This problem is further exacerbated as users increasingly depend on multiple devices such as PCs, laptops, and PDAs, some of which may frequently change their connectivity and bandwidth status, requiring explicit reasoning about the impact of disconnected operation.

In this paper, we present Wayfinder, a novel P2P file system that seeks to address the above limitations by presenting three synergistic abstractions: a global namespace that is uniformly accessible across connected and disconnected operation, content-based queries that can be persistently embedded into the global namespace, and automatic availability management. We choose to build around a file system paradigm for wide accessibility: data stored in files are easily accessible through a wide range of applications, including simple utilities such as `cat`, `grep`, and `awk`. Given this central paradigm, the three abstractions then serve to unify the fragmented views of data spread across multiple users and their devices, providing device-independent name *and* content addressing that naturally encompasses disconnected operation. Underneath the global data view, Wayfinder automatically manages the replication and placement of data to achieve specified availability targets.

In the remainder of this section, we describe the three abstractions mentioned above in more details. In the body of the paper, we describe Wayfinder's design and a prototype implementation; in particular, a key aspect of Wayfinder's design is that its abstractions are all implemented as queries against meta-data stored in an underlying P2P indexed storage layer called PlanetP [7]. Thus, we briefly describe PlanetP, how Wayfinder leverages PlanetP's indexing capabilities and query language to implement its exported abstractions, and how Wayfinder uses a light-weight distributed hash table (DHT) as a caching infrastructure to make its query-based design efficient. Finally, we close with some thoughts on the benefits that can be derived from enhancing PlanetP's query language, particularly as shared content are increasingly structured via XML.

Global namespace. Wayfinder constructs its global namespace by overlaying the local namespaces of individual nodes within a sharing community as shown in Figure 1. Each node's local namespace is called its *hoard* and consists of a directory structure and files stored in a local persistent storage system. The community may, at any point, split into multiple connected subsets, each with its own shared namespace, and later rejoin to recreate the entire global namespace. In essence, Wayfinder presents a shared view of all data stored across any set of connected nodes that expands and contracts smoothly on node arrival and departure.

Wayfinder adopts the above merging approach as opposed to today's mounting approach because it provides three important advantages. First, it provides users with a consistent browsing experience similar to the web but avoids the binding of names to specific devices. In particular, it allows any user to contribute content to any portion of the namespace by binding files to appropriate names within the local namespace of any of his devices; indeed, a user can modify any portion of the namespace by modifying, adding, and deleting files and directories provided that they have the necessary rights to do so[2]. This makes it possible for any Wayfinder community to collabora-

[2] Wayfinder implements a security model that allows a community to control write access to files and directories. A discussion of this model is beyond the scope of this paper, however. We refer the interested reader to [19].

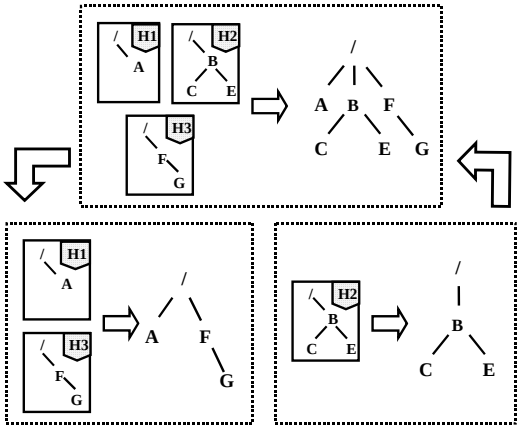

Fig. 1. *Wayfinder dynamically constructs a shared namespace across any set of connected devices by merging their local hoards. This figure shows 3 nodes originally being connected so that the shared namespace is the merged view of hoards H1 through H3. When the community is partitioned into 2 connected subsets, Wayfinder maintains a merged view for each subset. When the subsets reconnect, Wayfinder dynamically re-merges the shared namespace.*

tively organize shared information in a manner similar to the web-based DMOZ project (http://dmoz.org).

Second, it reduces the data management overheads by removing the need for users to explicitly reason about what replica resides on which device. As shall be seen, when a user accesses a file, Wayfinder will locate the latest version of that file within the connected community. Wayfinder also detects and automatically resolves conflicts that arise because of changes made during disconnected or partitioned operation. Wayfinder maintains sufficient information for users to manually resolve these conflicts if the automatic resolution is semantically incorrect.

Finally, a merging approach naturally encompasses partitioned and disconnected operation as shown in Figure 1. At the extreme, disconnected operation simply means that the namespace will include only the local hoard; thus, while the amount of accessible content may change, the manner in which users browse the namespace and access files does not.

Semantic directories. A key lesson from the web is that successful management and sharing of large volumes of data require both *browsing*, i.e., name addressing, and *content search*, i.e., content addressing. To date, however, browsing and content search have typically been viewed as two separate approaches for locating and organizing information. Wayfinder seeks to integrate these two approaches through the implementation of semantic directories [11, 12], which are search queries that are embedded into the persistent file system namespace.

Semantic directories provide a powerful organizational paradigm because they allow users to create a persistent namespace that automatically binds content to multiple browsing paths. For example, a file discussing content search in P2P file systems might be found through two completely different pathnames, with one leading to a query for "content search" and the other to a query for "P2P file systems," *without requiring users*

to explicitly create these name bindings. While individual users can of course pose either of these queries independent of the namespace, devising "good" queries is often a difficult art. In our lab, we often share good web queries with each other; e.g., "look this up using the following query." Embedding queries into the namespace will allow users to easily preserve and share good queries. Users will also benefit from each other' fine-tuning of search results; for example, each addition of a file that is relevant to but does not match a query benefits the next user that browses the semantic directory looking for relevant information.

As shall be seen, semantic directories are periodically reevaluated to reflect changes in the shared data collection, thus turning the file system namespace into an *active* organizational tool. Further, similar to the HAC file system [12], Wayfinder allows users to explicitly fine-tune the content of a semantic directory rather than having to manipulate the query until it returns the exact set of desired files. Finally, Wayfinder implements an approximation of the TFxIDF vector space ranking algorithm so that files inside a semantic directory can be ordered based on their relevance to the directory's query.

Automatic availability management. Providing high data availability is a fundamental aspect of a file system. Achieving high availability in P2P systems, however, can be quite difficult because of the extensive disconnection already mentioned. Wayfinder addresses this problem by automatically replicating data to achieve explicitly specified availability targets. Wayfinder continuously monitors and predicts the availability of nodes in a sharing community in order to make replication and placement decisions. In addition, we are exploring a novel user-centric availability model that addresses the combined problem of hoarding, that is, ensuring the presence of data on a specific device for disconnected operation, and availability, which is typically defined as the probability of successful access when connected to a server. Our ultimate goal is to support a unified metric where a file is available if it can be accessed, regardless of the accessing device's connectivity state. Wayfinder seeks to provide high user-centric availability, which works together with the dynamic global namespace to remove the need for users to explicitly reason about what replica resides on which device across connected and disconnected operations. This aspect of Wayfinder is still at the exploratory stage, however, and so will not be described further in this paper.

Design and Implementation Status. Wayfinder stores each node's hoard in the node's local file system and stores its meta-data in PlanetP, which is a P2P indexed data storage layer that supports a simple boolean query language for data retrieval. Wayfinder then uniformly constructs its file system namespace, locates specific files, and performs content searches by posing appropriate queries to PlanetP.

We have implemented a Wayfinder prototype that is sufficiently complete to support the transparent execution of common applications such as cvs, emacs, latex, gv, etc. on Linux. Currently, Wayfinder targets medium size communities of hundreds to several thousands of users as the common unit of social interaction. A good example of such a social group is the local P2P network already mentioned above that is comprised of several thousand students sharing approximately 25TB of data. Sharing within our laboratory (and department) is another example environment where we are deploying and using Wayfinder.

Our contributions include:

- The design and preliminary evaluation of a P2P file system that unifies name and content addressing in the context of a global browsable and writable namespace. Critically, our system supports content addressing and ranking without requiring any centralized indexing.
- The unification of two recent paradigms for building robust decentralized systems, distributed hash tables (DHT) [26] and replication using gossiping, to build a robust and efficient P2P file system.

2 Background: PlanetP

We begin by briefly describing PlanetP since Wayfinder uses it as a distributed data store for its meta-data. PlanetP is a toolkit that provides three building blocks for the construction of robust, medium-scale P2P applications: a gossiping module [7, 9], an indexed storage system and distributed query processing engine, and a lightweight active DHT. PlanetP's indexed storage system stores information as bindings of the form $\{k_1, k_2, ..., k_n\} \rightarrow o$, where k_i is a text key, o is an arbitrary object, and we say that $keys(o) = \{k_1, k_2, ..., k_n\}$. Stored objects are retrieved by specifying queries comprised of text keys combined using three operators, *and* ($\wedge$), *or* ($\vee$), and *without* ($-$). For example, a query ("cat" $\wedge$ "dog" $-$ "bird") would retrieve the set $\{o \mid (\{cat, dog\} \subseteq keys(o)) \wedge (\{bird\} \not\subseteq keys(o))\}$.

When a binding $\{k_1, k_2, ..., k_n\} \rightarrow o$ is inserted into PlanetP[3] at a particular node, PlanetP stores o in a persistent store local to that node, e.g., a BerkeleyDB database [25], and $k_1, k_2, ..., k_n$ in a two-level index. The top level of this two-level structure is a globally replicated key-to-node index, where a mapping $k \rightarrow n$ is in the index if and only if at least one binding $\{..., k, ...\} \rightarrow o$ has been inserted at node n. The second level is comprised of a set of *local* indexes, one per node, which maintains the key-to-object mappings for all bindings inserted at each node. The global index is currently implemented as a set of Bloom filters [4], one per participating node, and is loosely synchronized over time using the gossiping module [7].

To evaluate a query posed at some node n, PlanetP uses n's replica of the global index to identify target nodes that may contain relevant bindings. Then, PlanetP can either forward the query to all targets for exhaustive retrieval or to only a subset of targets that are likely to contain the most relevant objects. Contacted target nodes evaluate the query against their local indexes and return URLs and relevance rankings for matching objects to n. PlanetP's relevance ranking is computed using an approximation of the text-based TFxIDF vector space ranking algorithm [6].

To complement the gossip-based persistent indexed store, PlanetP also implements an active unreliable DHT. This DHT is active in that stored objects can execute on the hosting nodes but unreliable in that it may loose objects arbitrarily because nodes may leave (fail) without redistributing their portions of the DHT. There are two main expected use of this DHT: weak serialization of potentially conflicting operations and

[3] We often refer to the indexed data store as just PlanetP when the reference is clear within the surrounding context.

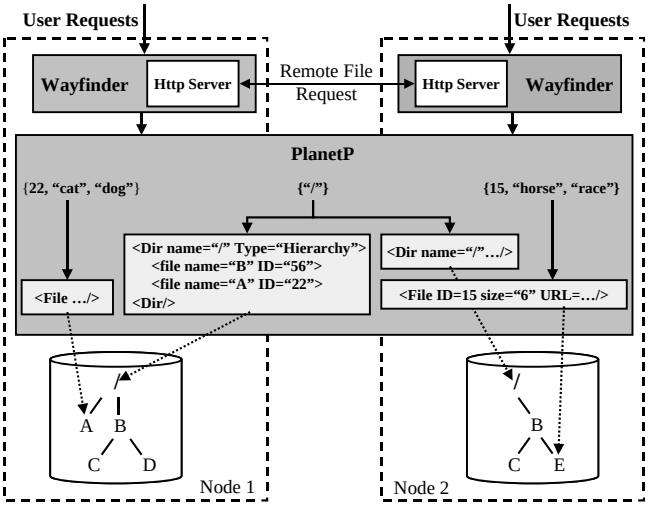

Fig. 2. *An overview of Wayfinder's architecture. Solid arrows inside PlanetP indicate bindings of keys to waynodes while dashed lines indicate implicit bindings of waynodes to file/directory replicas. Note that only a few waynodes are shown for clarity.*

caching of *soft* state such as those reconstructible from data in the indexed store to enhance performance.

We have shown that PlanetP's content ranking algorithm can achieve similar accuracy to a centralized implementation of TFxIDF [6]. In addition, PlanetP is extremely robust, even for highly volatile communities, and currently scales well to thousands of nodes [7] (hence the "medium-scale" label).

3 Files and Directories

We now describe how Wayfinder implements files and directories. As already mentioned, Wayfinder stores each node's hoard in the node's local file system and stores its meta-data in PlanetP (Figure 2). More specifically, each hoard is a sub-directory comprised of a portion of the files in the global namespace and the corresponding directory structure. Each file and directory can have many replicas across the community, with each replica described by a small data structure called a *waynode*. The set of all waynodes comprises Wayfinder's meta-data and is stored in PlanetP, where each waynode is bound to a set of content keys and a unique file or directory ID key. Wayfinder then constructs the global namespace, locates individual files, and performs content searches by posing appropriate queries to PlanetP. To make this query-based design efficient, Wayfinder caches query results in PlanetP's DHT. As shall be seen, Wayfinder only caches *soft state*, which can be arbitrarily lost at any point in time without affecting its correctness.

Files. Each Wayfinder file is identified by a unique identifier while each replica is described by a waynode. Each waynode contains a file ID, a version, a content hash, and

a URL indicating where the replica can be retrieved. Each waynode is stored in PlanetP, bounded to the replica's file ID and keys extracted from the replica's content. Each replica of a file */path/name* is stored at *path/name* in some hoard.

More specifically, suppose that the hoard of a node n is stored at */WFRoot* in n's local file system. When a file */path/f* is opened, Wayfinder first retrieves f's ID from the meta-data associated with */path* (see below) and queries PlanetP for all the waynodes of all replicas of f to compute the latest version and where replicas or diffs can be retrieved. To avoid this expensive operation in the future, Wayfinder caches the result in the DHT. Cache entries are active objects that can receive and process information about updates and so can persist in the cache indefinitely; in our current implementation, they are discarded after not having been accessed within some threshold period of time.

Then, if n does not have a local replica of f, Wayfinder retrieves a copy, stores it at */WFRoot/path/f*, and creates a new waynode for this replica and inserts it into PlanetP. The new waynode contains the same file ID and version number but has a new URL pointing to n's hoard. On the other hand, if n has an old version, Wayfinder updates the local copy to the latest version and updates the waynode to the new version number. Finally, Wayfinder completes the open on the local replica. Creation works similar to open except that Wayfinder generates a new file ID for the newly created file.

If f was opened for write, on closing, Wayfinder increments the replica's version number (in the waynode) and updates the meta-data object cached in the DHT for f if one exists. Wayfinder also computes a diff that contains all changes since the open and stores it in */WFRoot/path* together with *f*. (Of course, diffs are hidden from the user's view of the directory in the global namespace.) Diffs allow nodes to update replicas of large files without downloading the entire file. Diffs also allow Wayfinder to unroll changes as necessary to resolve write conflicts (see Section 4). Finally, Wayfinder schedules the file to be indexed in the background; if the file has already been indexed in the past, then Wayfinder can do an incremental index using just the diff. Once the index is completed, Wayfinder rebinds the waynode to the extracted content keys.

Directories. When a user opens or creates a file */path/f* at a node n, Wayfinder creates the directory *path* in n's hoard if it does not already exist. Thus, directories are replicated across nodes' hoards as well, although as shall be seen, they are only partially replicated. Directories are uniquely identified by their pathnames since directories with the same name are merged in the global view.

Each directory replica is represented by a waynode that stores all the name-to-replica bindings in the *local* hoard. That is, if a directory *path* in n's hoard contains two files, f_1 and f_2, then the waynode for */path* would contain two bindings, one binding the name f_1 to f_1's ID and one binding f_2 to f_2's ID. Each directory waynode w is inserted into PlanetP as a binding $\{$"*/path*"$\} \rightarrow w$. Then, to construct a global view of a directory */path* (for example, when the user does an *ls*), Wayfinder retrieves all waynodes bound to the key "/path" and merges their content.

We also cache directory views for efficiency. Similar to our caching of file meta-data, we choose to continuously update cached directory views; thus, whenever a node adds or deletes a file, if a view of the file's parent directory exists in the cache, then it updates the cache view to reflect the operation.

Semantic Directories. A semantic directory is one whose name maps to a content query [11]. In Wayfinder, `mkdir` creates a semantic directory if the first character of the name is a "$". Currently, a semantic directory's name just consists of a sequence of space-separated terms that are combined using the "and" operator to form the query. Each semantic directory has a set of attributes that can optionally be set to direct the ranking and display of matching files.

On creation, a semantic directory is populated with files within its scope that matches the directory's query. A file is defined to match a query if the keys that its waynodes, i.e., the waynodes of the replicas with the latest version, are bound to satisfy the query. Depending on its attributes, a semantic directory can be populated with all matching files or only a subset of highly ranked files. Wayfinder periodically reevaluates each semantic directory's query to refresh its content; the reevaluation period is a user-specifiable attribute of the directory.

When a user creates a semantic directory $/a/b$, if a is a regular directory, then the user has a choice of populating b with matching files from the entire file system (global scope) or only files contained in a (parent scope). If a is a semantic directory, however, then only parent scoping is allowed. Thus, a chain of three semantic directories $b/c/d$ would give three sets of files equivalent to the queries b, $b \wedge c$, and $b \wedge c \wedge d$.

Similar to [12], Wayfinder's semantic directories can be directly manipulated by users. That is, users can add files to or remove files from a semantic directory just like a normal directory. Files explicitly removed by a user are never brought back by a reevaluation although they can be added back explicitly. Likewise explicitly added files are never removed by reevaluation, even if their content do not match the directory's query.

Semantic directories are implemented as follows. When a node accesses a semantic directory, a replica is created in its hoard along with a waynode. The waynode is used to record explicit user manipulations of the semantic directory at that node, i.e., additions and deletions. On (re)evaluation, Wayfinder poses the directory's query to PlanetP and retrieves all waynodes that matches the query. Wayfinder also gathers all waynodes describing replicas of the directory. It then modifies the set of matching file by the union of the actions contains in the directory waynodes. Actions are ordered using logical timestamps; conflicting operations are resolved conservatively, favoring addition over deletion.

The result of the above evaluation is cached in memory until the next evaluation. When a file inside a semantic directory is accessed, a copy of it is downloaded to the hoard just as for a normal directory. If that file is later accessed through another pathname, or if a local replica already exists, Wayfinder only keeps one copy in the hoard and uses hard links to support accesses through the different pathnames.

4 Consistency

Wayfinder exports a weak consistency model similar to that of Bayou [20] for both directories and files to support partitioned operation—recall that Wayfinder continues to operate even when the sharing community is splintered into several disconnected parts. In this section, we describe this consistency model and its implications for users.

Files. Recall that when a user attempts to access a file f at some node n, n simply opens the latest version of f that it can find. This essentially implements a "single copy, any version availability" model [13]. Under partitioned operation, this model can lead to users seeing stale data and conflicting non-concurrent writes because of incomplete hoarding within n's partition or recent writes outside of n's partition. (Note that these problems can arise even when the entire community is connected: when cached entries in the DHT are lost, gossiping delays may give rise to inconsistent views of the global index, which in turn may lead to inconsistent actions. These inconsistencies are subsumed by those arising from partitioned operation, however, and so are dealt with similarly.)

The above inconsistencies are *inherent* to any system that supports partitioned operation. Wayfinder's replication approach (not described here) reduces the probability of accessing stale data. To address write conflicts, Wayfinder maintains a version vector in each waynode, where a new version extends the vector with a monotonically increasing number and the ID (a hash of a public key) of the writer. Then, when Wayfinder detects write conflicts, it imposes an arbitrary but deterministic and globally consistent ordering on the changes. This allows nodes to resolve conflicts without the need to reach a communal consensus.

For example, suppose Wayfinder detects two waynodes for the same file with conflicting versions [(x,1)(y,2)] and [(x,1)(z,2)]. Further suppose that $y < z$ according to their integer values. Wayfinder would then apply the diff between [(x,1)] and [(x,1)(z,2)] to [(x,1)(y,2)] to get the version [(x,1)(y,2)(z,2)]. To address the cases when this resolution is semantically incorrect—although often, similar to CVS conflict resolution, this automatic resolution may be correct—Wayfinder allows users to manually merge diffs to create a new, semantically correct version. Continuing the example, Wayfinder allows a user to create a new version [(x,1)(y,2)(z,2)(u,3)] by providing the [(x,1)] version using diff rollback and the two conflicting diffs.

Directories. Wayfinder also supports a "single copy availability" model for directory accesses. Suppose a user at some node n attempts to access a directory */path/dir*. This access will succeed if any node in n's partition has a replica of */path/dir*. For similar reasons as above, this model can cause users to not see bindings that actually exist, see bindings that have been deleted, and create conflicting bindings. Since replicating files involve replicating their ancestor directories, our replication approach also reduces the probability of incomplete views. To resolve conflicting bindings when creating a directory view, Wayfinder renames the bindings in the DHT cache entry and notes this rebinding. When a user attempts to access a file through the renamed binding, Wayfinder notifies the user of the conflict so that a permanent rebinding can be affected.

To delete a binding */path/f*, Wayfinder unlinks *path/f* in the local hoard, removes f from the cached entry of */path* in the DHT, and publishes a delete notification to PlanetP. Whenever a node accesses */path*, it will see the delete and remove its own local replica if it has one. Each node also periodically looks for delete notices and removes any corresponding local replicas. Delete notifications are discarded after an expiration period currently set to four weeks. Thus, it is possible for a node that was offline for longer than this period to bring back a copy of a deleted file when it comes back on-line.

Modified Andrew Benchmark				
Phase	Linux NFS	JNFSD	Wayfinder: 1 Node	Wayfinder: Worst Case
1	0.02 s	0.04 s	0.04 s	0.10 s
2	0.18 s	0.37 s	0.82 s	1.51 s
3	1.03 s	0.82 s	0.85 s	1.08 s
4	0.84 s	1.58 s	1.64 s	1.82 s
5	2.09 s	3.13 s	3.30 s	3.49 s
Total	4.16 s	5.94 s	6.65 s	8.01 s

Table 1. *Results of the Modified Andrew Benchmark using the Linux NFS, original JNFSD and the JNFSD linked with Wayfinder running in isolation and connected to a large community of nodes.*

To delete a directory, Wayfinder deletes all files within the directory as described above and deletes the directory itself from the hoard. When processing delete notifications, a node also recursively delete ancestor directories if the deleted binding was the last in that directory. This implementation has two implications. First, since deleted files can reappear, so can deleted directories. Second, deleting the last binding in a directory effectively deletes that directory as well.

Finally, since we depend on nodes to update cached directory entries in the DHT to reflect changes, these entries may become stale when a node goes offline, modifies its local hoard, then returns. To address this problem, cached entries are automatically discarded after an expiration period. Also, when a node rejoins an online community, it lazily walks through its hoard and updates any stale cached entries. When two connected subsets join, cached entries for the same directory are merged.

5 Performance

We now consider the performance and robustness of a prototype implementation. Results presented here are preliminary since we are just starting to use Wayfinder on a daily basis.

Our prototype is written in Java and uses a modified JNFSD server [16] to export its services as a locally mounted user-level NFS system. All experiments are performed on a cluster of PCs, each equipped with an 800MHz PIII processor, 512MB of memory, and a 9GB SCSI disk. Nodes run Linux 2.2.14 and Sun's Java 1.4.1_2 SDK. The cluster is interconnected by a 100Mb/s Ethernet switch.

Each Wayfinder node caches meta-data retrieved from the DHT in local memory for 10 seconds. In our current setup, this reduces the impact of accessing the DHT through Java RMI, which requires on order of 2.5ms for a single RPC. When Wayfinder is used by communities connected over the Internet, this caching reduces the impact of communication over the WAN. Note that this caching is similar to caching done by the Linux NFS client (3–30 seconds), although Linux has a more sophisticated policy of when to disregard the cache.

Andrew Benchmark. Table 1 shows the running time for the Modified Andrew Benchmark [15] for Linux NFS, the unmodified JNFSD, and Wayfinder. The benchmark con-

sists of five phases executed by a single client: (1) create a directory structure, (2) copy a set of files into the directory structure, (3) stat each file, (4) grep through the files, and (5) compile the files. In all cases, the NFS server and client ran on the same machine for comparison against when Wayfinder is running on a single node. For Wayfinder, "Worst Case" reflects performance for the hypothetical scenario where the community is very large so that each access to the DHT requires a message exchange. Since all operations are performed on a single client, these remote DHT accesses are the most significant source of overhead for this benchmark.

Observe that Wayfinder imposes little overhead when the workload is not entirely comprised of file system operations. In particular, Wayfinder imposes insignificant overheads for phases 4 and 5, when the client is grepping and compiling, respectively. Phase 1 and 2 impose higher performance penalty, particular phase 2 where each copy requires Wayfinder to compute a diff and to synchronously flush the corresponding waynode from the local cache, forcing a remote DHT update. Currently, the computation of a diff involves copying the entire file at an open which we plan to optimize in the future by implementing copy-on-write. Phase 3 benefits from the cache footprint resulting from phase 2 and so imposes only a modest amount of overhead.

We thus conclude that while Wayfinder does impose visible overheads on basic file system operations. These overheads are quite acceptable given that the prototype is a largely un-tuned Java program. We also observe that the Andrew Benchmark gives the worst case scenario for Wayfinder: all operations are performed at a single client and so gives no measure of Wayfinder's effectiveness for collaborative workloads.

Scalability and Robustness. We now show the advantage of Wayfinder's dual nature, using gossiping for robustness to failures and caching in the DHT for scalable performance. In this experiment, we turn off the caching at the local node to force accesses to use either the DHT or PlanetP's retrieval.

Figure 3(a) plots the time required for a single node to perform a complete traversal of a namespace, e.g., doing an "ls -R" vs. community size with, and without, the use of caching in the DHT. The namespace is a complete trinary directory tree of depth 3, giving a total of 41 directories with each directory containing 1 file. Each node hoards the entire namespace.

As expected, the scan time without caching in the DHT grows linearly with community size since computing each directory view requires contacting all nodes. With caching, however, the scan time rises only slightly with community size as more and more cached entries are stored at remote nodes; this curve has an asymptote, however, corresponding to the cost of a network access per directory access.

On the other hand, Figure 3(b) shows Wayfinder's robustness to loss of DHT data. In this experiment, we run a sequence of scans and, in two instances, we simulate node crashes by causing 2 and 4 nodes, respectively, to drop all of their DHT entries and leave the community. The scan is performed over a similar (albeit slightly smaller) directory structure as before but where each file is replicated only twice so that each crash leaves some files with only one replica. Observe the rise in scan time right after the simulated failures because some directory views had to be reconstructed. These scans correctly recreated all views, however, and re-cached them.

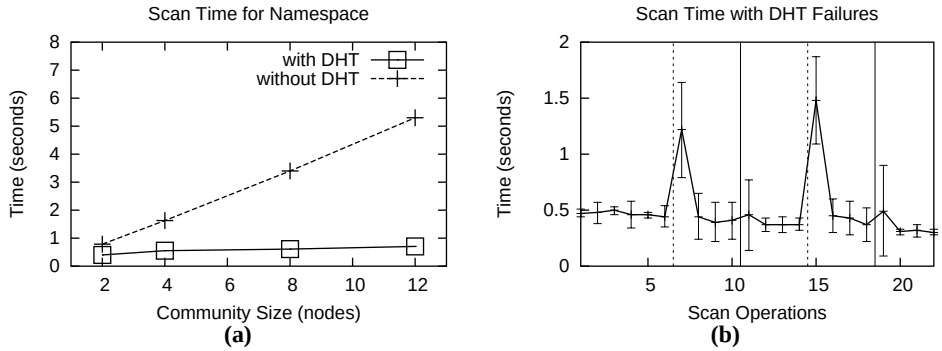

Fig. 3. *(a) Time required to scan a namespace plotted against community size. (b) Scan time in the presence of node failures; x-axis is a sequence of scans, dotted vertical lines indicate node failures, and solid vertical lines indicate the return of failed nodes. 2 nodes failed after scan 6 while 4 nodes failed after scan 14. The vertical bar gives the standard deviation for scan time across 10 samples.*

6 Related Work

Several previous efforts have investigated the provision of a global namespace spanning multiple file and publishing systems [18, 21]. However, these efforts were more focused on providing individual users with logical namespaces that span multiple file systems rather than a communal namespace that can be collaboratively organized. The Federated File System (FFS) [27] is probably closest to our work with respect to namespace construction but FFS is targeted specifically for a cluster rather than a P2P community.

The Semantic File System [11] introduced the concept of semantic directories, which was further developed in the HAC File System [12]. Wayfinder implements this abstraction in the context of P2P systems. Wayfinder also introduces content ranking within semantic directories.

Many projects have recently explored P2P file systems. However, to our knowledge, none of these systems have considered content search and ranking. Further differences are as follows. The Secure Read-Only File System [10] and the Cooperative File System [8] are read-only publishing file systems. Farsite [1], a general read/write server-less file system, shares many common goals with our work. However, Farsite targets a corporate environment with significantly different characteristics than our target environments and so its design and implementation is significantly different from Wayfinder. Oceanstore [22] and Pangaea [23] are more concerned with extreme scaling than Wayfinder and so their designs and implementations are also significantly different than Wayfinder. Ivy [17] is a P2P file system that stores its data blocks in a DHT. This approach is fundamentally different than ours and may lead to unacceptably high data movement for highly dynamic communities [5, 3]. Finally, Bhagwan et al. have considered automatic availability management in the context of a storage system built around a DHT [3]. This system's replication approach is similar to that of Wayfinder (although

this aspect of our work is not described here) but it does not target disconnected operation nor content search and ranking.

Several projects have addressed the enhancement of data management capabilities in P2P networks. For example, Harren et al. are exploring the introduction of complex query languages to P2P systems, particular in the context of systems built on DHTs [14]. The Piazza project is seeking to support the sharing of heterogeneous, semantically rich data in P2P systems [29]. These efforts are complementary to ours in that more powerful querying languages and engines on top of semantically rich data will likely increase Wayfinder's capability for providing a convenient yet powerful collaborative information sharing environment.

Tang and Dwarkadas recently investigated content search and ranking for DHT-based P2P systems [28]. This work is similar to PlanetP's content search and ranking albeit it targets a different underlying building block for P2P systems.

7 Conclusions and Future Work

We have presented Wayfinder, a novel P2P file system that seeks to unify publishing, searching, and collaborative organization within the context of a file system to better support the needs of medium-sized content sharing networks. Specifically, we have described two of the three critical abstractions exported by Wayfinder: a global namespace that merges devices' local namespaces into a unified view and semantic directories that allow the namespace to actively organize the shared information. We have shown how Wayfinder implements these abstractions on top of a P2P indexed data store; specifically, Wayfinder stores all of its meta-data in this data store. Wayfinder then constructs the global namespace, locates specific files, and performs content searches by posing appropriate queries to the underlying storage system. We have also described how to make this query-based design efficient by caching the query results in an unreliable DHT. Finally, we have given preliminary performance measurements collected from a prototype to show that we can achieve reasonable performance.

We are currently in the process of deploying Wayfinder inside our lab (and hopefully within our department) for actual use to evaluate Wayfinder's impact on everyday data management tasks. We are also pursuing two directions of future work. First, we are exploring the usefulness of Wayfinder's semantic directories as well as the opportunity to improve its content ranking capabilities by observing users' access patterns and explicit manipulations of semantic directories. For example, if a user explicitly adds a file to a semantic directory or accesses a lowly ranked file frequently, we might be able to use the content of these files to "improve" the query through keyword expansion.

Second, we are exploring how PlanetP can support a more complex query language. Harren et al. [14] have pointed out that such an effort can be rewarding as it can significantly enrich ways in which applications like Wayfinder can help users locate and manage shared content more effectively[4]. Concurrently, we will also explore the storage of Wayfinder's files as well as its meta-data in PlanetP. In essence, we seek to explore a

[4] Interestingly, PlanetP already supports part of the API identified by Harren et al. as being useful for implementing complex query languages in P2P systems.

content management and sharing system where coherent views, e.g., files and directories, are just results of appropriate queries posed on an underlying "sea of data." Such a hybrid system is powerful in that it supports a simple file system API yet also provides the benefits of a powerful underlying data management system. For example, in our current system, a semantic directory can be thought of as a dynamic "attraction" point, where new content entering the system related to a set of keywords will automatically be listed. Yet, applications accessing such a directory simply opens, reads, and writes the directory without having to know anything about the underlying data management capabilities. One can imagine equivalent *semantic* files that can serve to attract snippets of information; current PDA address books and calendars are examples of how such semantic files might be useful.

Acknowledgements. We thank Kien Le for developing Wayfinder's content indexer. We thank the reviewers for their thoughtful comments.

References

1. A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.
2. R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2003.
3. R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. M. Voelker. Total Recall: System Support for Automated Availability Management. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, June 2004.
4. B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, July 1970.
5. F. M. Cuenca-Acuna, R. P. Martin, and T. D. Nguyen. Autonomous Replication for High Availability in Unstructured P2P Systems. In *Proceedings of the Symposium on Reliable Distributed Systems (SRDS)*, Oct. 2003.
6. F. M. Cuenca-Acuna and T. D. Nguyen. Text-Based Content Search and Retrieval in ad hoc P2P Communities. In *Proceedings of the International Workshop on Peer-to-Peer Computing*, May 2002.
7. F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*, June 2003.
8. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, Oct. 2001.
9. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, 1987.
10. K. Fu, M. F. Kaashoek, and D. Mazires. Fast and secure distributed read-only file system. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, Oct. 2000.
11. D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O. Jr. Semantic File Systems. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, Oct. 1991.

12. B. Gopal and U. Manber. Integrating Content-Based Access Mechanisms with Hierarchical File System. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, Feb. 1999.
13. R. G. Guy, J. S. Heidemann, W. Mak, T. W. Page, Jr., G. J. Popek, and D. Rothmeir. Implementation of the Ficus Replicated File System. In *Proceedings of the Summer USENIX Conference*, June 1990.
14. M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, , and I. Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Apr. 2002.
15. J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems (TOC)*, 6(1), Feb. 1988.
16. Java nfs server. http://members.aol.com/_ht_a/ markmitche11/jnfsd.htm, Oct. 2002.
17. A. Muthitacharoen, R. Morris, T. Gil, and I. B. Chen. Ivy: A Read/Write Peer-to-Peer File System. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.
18. B. C. Neuman. The Prospero File System: A Global File System Based on the Virtual System Model. In *Proceedings of the Workshop on File Systems*, May 1992.
19. C. Peery, F. M. Cuenca-Acuna, R. P. Martin, and T. D. Nguyen. Wayfinder: Navigating and sharing information in a decentralized world. Technical Report DCS-TR-534, Department of Computer Science, Rutgers University, Oct. 2003.
20. K. Petersen, M. Spreitzer, D. Terry, and M. Theimer. Bayou: Replicated Database Services for World-Wide Applications. In *Proceedings of the Conference on Operating Systems (SIGOPS)*, Sept. 1996.
21. H. C. Rao and L. L. Peterson. Accessing Files in an Internet: The Jade File System. *Software Engineering*, 19(6), June 1993.
22. S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The oceanstore prototype. In *Proceedings of the Conference on File and Storage Technologies (FAST)*, 2003.
23. Y. Saito and C. Karamanolis. Pangaea: a symbiotic wide-area file system. In *Proceedings of the Conference on Operating Systems (SIGOPS)*, Sept. 2002.
24. S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking (MMCN)*, Jan. 2002.
25. S. Software. Berkeley DB. http://www.sleepycat.com/.
26. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the Conference on Data Communications (SIGCOMM)*, Aug. 2001.
27. L. I. Suresh Gopalakrishnan, Ashok Arumugam. Federated File Systems for Clusters with Remote Memory Communication . Technical Report DCS-TR-472, Department of Computer Science, Rutgers University, Dec. 2001.
28. C. Tang and S. Dwarkadas. Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, June 2004.
29. I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau, and P. Mork. The Piazza Peer Data Management Project. In *Proceedings of the Conference on Management of Data (SIGMOD)*, June 2003.