Wayfinder: Navigating and Sharing Information in a Decentralized World

Christopher Peery, Francisco Matias Cuenca-Acuna, Richard P. Martin, Thu D. Nguyen
{*peery, mcuenca, rmartin, tdnguyen*}*@cs.rutgers.edu*
Department of Computer Science, Rutgers University
110 Frelinghuysen Rd, Piscataway, NJ 08854
Department of Computer Science Technical Report DCS-TR-534

Abstract

Technology trends are combining to fundamentally raise the information management burden using today's storage systems. Currently, users manually track file replicas across different systems, such as file systems, Web servers, and CVS trees, all using different names and spanning numerous devices. In addition, users must reason about the impact of disconnected operation because devices frequently change their connectivity and bandwidth status. In this paper, we describe Wayfinder, a peer-to-peer file system designed to reduce the increasing burden of information management by directly supporting the needs of dynamic medium-sized collaborative communities within a familiar file system interface. Wayfinder achieves this goal by exporting three critical abstractions: a unified namespace built by merging multiple directory views, content addressing via semantic directories, and availability conscious replication. We demonstrate the feasibility of our approach by measuring the performance 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 systems can be quite frustrating. First, these systems typically export a publishing paradigm where each individual's published content is stored in a private repository that is read-only to other users. This makes content location through browsing very time consuming because each repository is typically organized in a completely different way. Second, content search is typically too primitive to be useful except in cases where the community is externally indexed by a web search engine, e.g., Google's indexing of Usenet. Finally, it is impossible to reason about data availability when content is hosted by community members because of extensive disconnected operation [1, 24].

Managing shared content is also difficult, particularly when users participate in multiple networks, as users must manually track content replicas across multiple publishing 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 in their connectivity and bandwidth status, requiring explicit reasoning about the impact of disconnected operation.

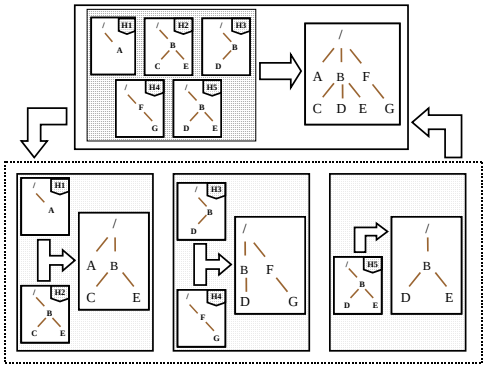

Figure 1: *Wayfinder's shared namespace is constructed by merging local directory hierarchies (hoards) across connected nodes. This figure shows 5 nodes originally being connected so that the shared namespace is the merged view of hoards H1 through H5. When the connectivity changes and the community separates into three connected subsets, Wayfinder maintains a merged view for each subset. When the subsets reconnect, Wayfinder dynamically re-merges the shared namespace.*

Together, the above problems will fundamentally limit users' ability to manage the ever growing volume of shared data and to develop richer sharing patterns. Thus, in this paper, we present Wayfinder, a novel P2P file system that advances the state of the art by unifying publishing, search and storage. In particular, Wayfinder addresses the above limitations by exporting three synergistic abstractions: a unified global namespace, content addressing, and availability-conscious replication.

Unified Namespace. To support sharing, Wayfinder constructs a universal shared namespace across a connected community. This shared namespace is formed by overlaying the local namespaces of participating nodes 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 file system. The community may, at any point, split into multiple connected subsets, each with its own shared namespace, and later rejoin to recreate the communal 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's communal namespace aids users in three ways. First, it eases content sharing by presenting a familiar file system model yet publishes any file created or updated inside the hoard of a device. Second, it allows users to build on each other's organizational efforts. Finally, it aids a user in managing his data set across multiple devices by removing the need to explicitly reason about what replica resides on which device.

Content Addressing. Wayfinder supports content addressing by providing semantic directories [8, 9], which represent search queries and are populated with their query results. Semantic directories allow users to embed a multi-dimensional content-based navigation structure within the normal directory hierarchy. For example, a file discussing P2P file systems might be found through a semantic directory querying for "P2P" or through one querying for "file system." More over, this structure actively organizes files as their content change since semantic directories are periodically reevaluated. Similar to [9], 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. Search results can also be content-ranked to assist users in finding the most relevant files.

Availability-Conscious Replication. Similar to previous systems that support disconnected operation, Wayfinder is faced with the dual issues of hoarding and availability. However, Wayfinder's explicit support for partitioned operation blurs the distinction between these issues: in a connected subset, is successful access an availability issue or a hoarding issue? 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.

In this paper, however, we first consider the constrained problem of ensuring file availability when nodes are connected to a *core* set of peers that are rarely partitioned. Specifically, availability is defined as the probability of finding a file at any point in time within the connected subset containing the core. As an example, the core set of our laboratory's Wayfinder community is comprised of our lab server and the faculty members' PCs. (Note that the core set only serves to define the connected state; they *are not* required to serve as highly available replication stores.)

Wayfinder implements the availability-conscious replication (ACR) approach previously introduced by Cuenca-Acuna et al. [3] to address the above availability problem. ACR has two important characteristics. First, it supports an explicitly non-uniform availability model, where files can have different target availabilities. Second, rather than maintaining a fixed number of replicas per file, it periodically adjusts the number of replicas depending on an explicit estimate of each file's availability. Both these characteristics are important for P2P systems, where maintaining uniform file availability through a fixed number of replicas is often impractical [3].

Cuenca-Acuna et al. have shown that ACR can achieve practical availability levels, e.g., 99.9%, in challenging P2P environments. ACR also scales gracefully with available resources, thus providing an interesting continuum. The availability model can range from one with extensive non-uniformity, similar to today's P2P file sharing systems, or almost completely uniform similar to server-based file systems. Where in the range a community exists depends on the amount of storage, placement, and availability of server-like nodes.

Design and Implementation. Wayfinder is really a meta file system in that it stores files in a node's hoard on any local file system and its meta-data in a lower layer called PlanetP [5]. Currently, Wayfinder targets medium size communities of tens to hundreds of users as the common unit of social interaction. A good example of such a social group is a local P2P network of several hundred students sharing 6TB of data. Sharing within our laboratory (and department) is another example environment where we plan to actually deploy and use Wayfinder. Finally, Wayfinder implements a simple security model that uses public key encryption to control writes. We will not describe Wayfinder's security model further here, however, because of space constraints; we refer the interested reader to [19].

Our contributions include:

- The design and evaluation of a file system that unifies name and content addressing within a dynamic navigational structure targeted to sharing and collaborative data management. Our system supports content addressing and ranking without requiring any centralized indexing.
- Introducing a configurable availability model that spans the fixed availability model of current file systems to the highly non-uniform models of current P2P systems.
- Unify the use of two recent paradigms for building robust decentralized systems, DHT and replication using gossiping, to build a P2P file system that supports partitioned operation. We discuss this synergistic design further in Sections 2 and 3.

2 Background: PlanetP

We begin by briefly describing PlanetP since Wayfinder uses it as a distributed data store for its meta-data. Essentially, PlanetP is a multidimensional indexed P2P data store. Members of a PlanetP data store share information by *publishing* bindings of the form $\{k_1, k_2, ..., k_n\} \rightarrow s$, where k is a text key and s an arbitrary object (although in Wayfinder, s is always a small XML document). We say that $keys(s) = \{k_1, k_2, ..., k_n\}$. Published documents are then retrieved by specifying a query comprised of a set of keys combined using three operators, *and* ($\wedge$), *or* ($\vee$), and *without* ($-$). For example, a query ("cat" $\wedge$ "dog" $-$ "bird") would retrieve the set $\{s \mid \{cat, dog\} \subseteq keys(s) \wedge \{bird\} \not\subseteq keys(s)\}$.

When a node publishes a binding $\{keys\} \rightarrow s$, PlanetP stores s in a local persistent data store and $\{keys\}$ in a two-level index[1]. 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 n has published at least one object s where $k \in keys(s)$. We call this first level index the *global* index. The second level is comprised of a set of independent *local* indexes, one per node, that maintains the set of key-to-object bindings for all bindings published at that node. The global index is currently implemented as a set of Bloom filters [2], one per node that summarizes the set of unique keys in that node's local index.

PlanetP uses gossiping to replicate and weakly synchronize the global index and a membership directory. In particular, each member periodically tells a randomly selected peer about changes to the replicated data structures so that changes eventually diffuse to all nodes. In the absence of changes, however, PlanetP throttles the gossiping rate so that bandwidth use quickly becomes negligible.

PlanetP evaluates a query by using the global index to identify the set of nodes that contain relevant bindings and passes the query to these nodes. The target nodes then evaluate the query against their local indexes and return URLs for matching objects to the querier. PlanetP can contact all targets in order to retrieve an exhaustive list or a ranked subset to retrieve only the most relevant objects [4].

To complement the gossip-based persistent indexing infrastructure, PlanetP also implements a lightweight, unreliable DHT as a distributing caching and fast rendezvous service. The DHT may loose data arbitrarily, however, because nodes may leave (fail) without redistributing their portion of the DHT. Thus, all data stored in the DHT must be reconstructible from the persistent store.

Finally, with respect to performance, we have previously shown that PlanetP can scale to thousands of nodes [5], which is more than sufficient for Wayfinder's current target community sizes.

[1]For simplicity, we assume that there is only one instance of PlanetP (and Wayfi nder) running on each node in our discussion. Our implementation supports many instances running on each node, each instance accessible by many users.

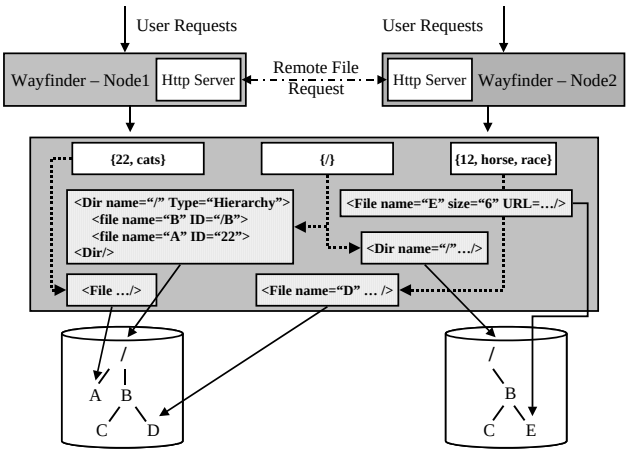

Figure 2: *An overview of Wayfinder's architecture. Dashed lines represent key bindings while solid lines represent Wayfinder's mapping of waynodes to local files and directories.*

3 Files and Directories

We now describe how Wayfinder implements files and directories. Figure 2 gives an overview of Wayfinder's architecture. Broadly, Wayfinder stores each user's hoard in a local file system and its meta-data in a PlanetP data store. Each file and directory can have many replicas. Each replica of a file or directory is described by a small data structure called a *waynode*. Each waynode is bound to content keys and Wayfinder assigned IDs. Wayfinder can then retrieve waynodes using a combination of content or ID searches.

Files. Each replica of a file */path/name* is stored at *path/name* in some hoard. 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. Waynodes are implemented as XML snippets and are stored in PlanetP[2]. They are bounded to the file ID and keys extracted from the replica's content.

To illustrate this design, 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 retrieves f's ID from the meta-data associated with */path* (see below) and queries the DHT for f's latest version and a set of URLs where replicas or diffs can be downloaded. If this query fails, then Wayfinder queries PlanetP for all the waynodes of all replicas of f to

[2]Typically, PlanetP is used to denote just the indexed persistent store; we always explicitly state when we use the DHT to cache information.

compute the latest version and where replicas or diffs can be retrieved. Wayfinder then caches the result in the DHT for future accesses to f. (Cached entries are discarded after they have not been accessed for some expiration period.)

Then, if n does not have a local replica of f, Wayfinder retrieves a copy, stores it at */WFRoot/path/f*, and creates and publishes a new waynode for this replica. The new waynode contains the same file ID and version number but has a new URL pointing to n's hoard. If n has an old version, Wayfinder updates the local copy to the latest version, re-publishes the waynode with the new version number, and updates the meta-data cached in the DHT. 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, republishes its waynode, and updates the meta-data 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 5). 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 waynode w is published with the binding $\{$"*/path*"$\} \rightarrow w$.

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. To avoid repeatedly contacting a large number of nodes to construct a view of the same directory, we cache the merged result as soft state in PlanetP's DHT.

In caching directory views, however, we are faced with what to do when the view changes (e.g. files are added or deleted from the directory). One option is to allow cached views to become out-of-date but discard them after some amount of time to limit inconsistencies. We chose the opposite approach of continuously updating cached views and only discarding a view if it has not been accessed for some time. Thus, whenever a node adds, deletes, or writes 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 [8]. 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.

On creation, a semantic directory is populated with all 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. The set of matching files can be ranked using PlanetP's relevance ranking if desired. Wayfinder periodically reevaluates each semantic directory's query to refresh its content; the reevaluation period may be specified by the user.

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 [9], Wayfinder's semantic directories can be directly manipulated by users. That is, users can add files to, or remove 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 waynodes by the union of the actions contains in the directory waynodes. Actions are ordered using logical timestamps; conflicting operations are resolved conservatively, favoring adding over deletion.

The result of the above evaluation is cached in memory until the next evaluation. When a file inside a semantic di-

rectory 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 Availability

In this section, we describe how Wayfinder replicates files to increase their availability. Recall that we currently define availability as the probability of finding a file within the connected subset surrounding a core set of nodes. Given this metric, Wayfinder exports a novel non-uniform availability model by implementing an adaptation of Cuenca-Acuna et al.'s availability-conscious replication (ACR) approach [3].

ACR assumes that each file has an explicit availability target that is either specified by the user or inherited by default from an ancestor directory. ACR also assumes that nodes periodically advertise their past availability, computed as a running average of time connected to vs. disconnected from the core. Given these measures, each node n is then independently responsible for maintaining the availability of files in its hoard, pushing additional replicas of a file whenever it estimates the file's availability to be below the target. This autonomous approach allows nodes to act independently, yet achieve the desired coherent global effect because information shared through PlanetP allows all nodes to compute similar availability estimates. In essence, Wayfinder will continuously monitor files with high availability targets since these files will either be replicated on highly available nodes or on many less available nodes. On the other hand, Wayfinder monitors files with low availability targets less carefully since either less nodes or less available nodes are maintaining these files' availability.

More specifically, each node n periodically chooses a random file f from its hoard, computes the set of peers that contain a replica of f using PlanetP's global index, and estimates f's availability assuming that peers' individual availabilities are uncorrelated. This estimate includes peers that are currently offline under the assumption that they will return later with their hoard intact. (Of course, hoards will change, in which case, everyone's copy of the global index will eventually be updated, and the availability estimates will change accordingly.) If the estimated availability is below the target availability, then n attempts to push a replica of f to a randomly chosen online peer that is not currently hoarding f. Otherwise, n contacts a randomly chosen peer that is currently hoarding f and ensures that both nodes have the latest version of f as known to them. This latter action ensures that a file's estimated availability (eventually) reflects the availability of the latest version.

When a node m receives a replication push for f, it accepts f if it has excess space in its hoard. If m's hoard is full, it can reject the push or evict files to free up space. Specifically, m chooses the necessary victims using a weighted random selection, where the weights are computed from files' estimated availability. m should already have availability estimates for its hoarded files from the above pushing process and so only needs to compute f's estimated availability.

Eviction brings up an interesting conflict: a Wayfinder node should not evict a file g, even if g is over-replicated, if some user is likely to access g during a period of disconnected operation. Currently, Wayfinder implements the simple policy of never evicting a file that some local user previously accessed and so pulled to the local hoard. This policy is limited in that most hoards will eventually fill up with files no longer useful in near future periods of disconnected operation. Thus, we are currently exploring a value function that allows Wayfinder to better balance between hoarding and availability needs.

5 Consistency

Wayfinder exports a weak consistency model for both directories and files to support partitioned operation. 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 [10]. 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, however, reduces the probability of accessing stale data when a node is connected to the community core. As already mentioned, we are currently exploring how to ensure availability during disconnected operation. Finally, 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 determin-

istic 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, 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.

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.

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

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

6 Performance

We now consider the performance and robustness of a prototype implementation. This prototype is written in Java and uses a modified JNFSD server [13] 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 [12] for Linux NFS, the unmodified JNFSD, and Wayfinder. The benchmark consists 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 sce-

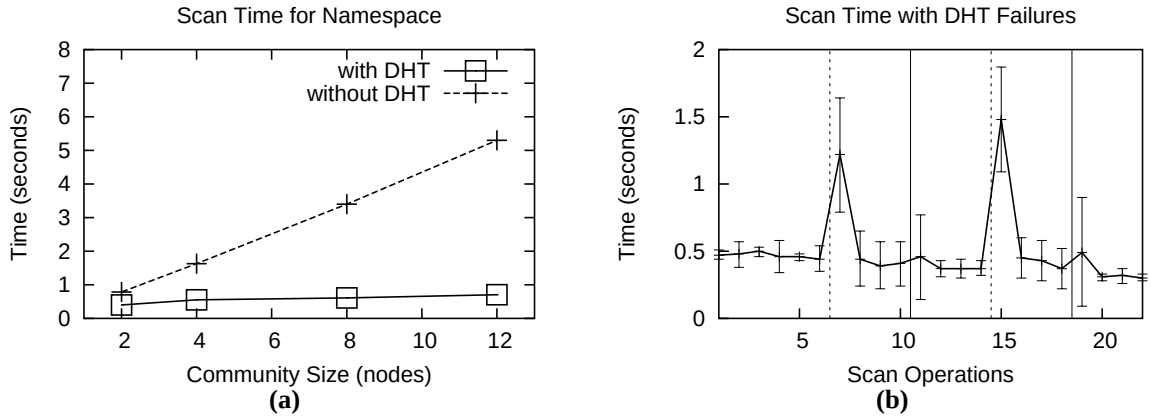

Figure 3: *(a) Time required to scan a namespace plotted against community size. (b) Scan time in the presence of node failures; x-axis is a sequence of scans, dotted vertical lines indicate node failures, and solid vertical lines indicate the return of failed nodes. 2 nodes failed after scan 6 while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

nario 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 phase 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. 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, which uses 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 4, giving a total of 121 directories, and each directory contains 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.

Space and Bandwidth. Finally, we consider the space and bandwidth overheads of maintaining Wayfinder's metadata. We have previously shown the space overheads of PlanetP's global index to be quite modest [5]. For example, if the entire TREC collection [11] (944,651 documents, 256,686,468 terms, 592,052 unique terms, 3,428.41 MB)

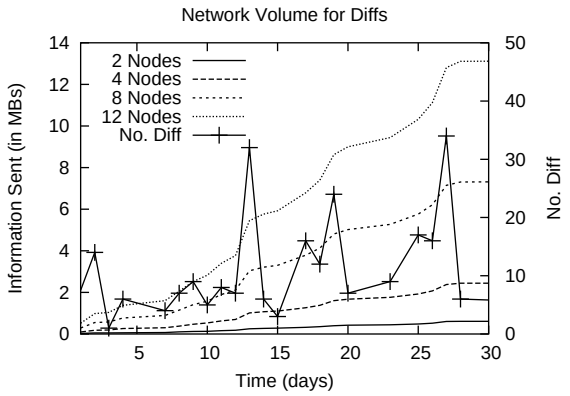

Figure 4: *The cumulative network volume in MB required to propagate diff information in different sized communities and the number of updates generated per day for a paper taken from the PANIC-Lab CVS Repository over the lifetime of the paper.*

were uniformly distributed across a community of 1000 nodes, giving an average of almost 20,000 unique keys per node, the global index would occupy about 16 MB of storage, or 0.5% of the collection. By comparison, when we indexed the files of 2 heavy users in our lab, including all files with extensions html, pdf, ps, tex, and txt, we found 26,000 keys for one and 353,000 unique keys for the other. The latter, however, was storing a portion of the Gutenberg project (http://www.gutenberg.net/index.shtml), which consists of a large number of e-books in several languages. Thus, we can expect the global index of a modest-sized community to be in the range of several to tens of MB. We did not measure the storage required for the local indexes as we did not implement any of a number of well-known optimizations to limit their sizes.

Finally, to give an idea of network usage, we estimate the messaging volume required to keep the global indexes of a community synchronized. This is the most significant of the messaging overheads required by Wayfinder to maintain its meta-data. In this experiment, we use the files and updates for one paper (somewhat larger than this one) kept in our CVS repository. Four people collaborated on the authoring of this paper over a period of approximately 30 days.

Figure 4 shows the cumulative network volume used if each CVS commit of a file was assumed to correspond to an open/modify/close edit cycle. We collected this data using a combination of micro-benchmarking of the prototype and modeling using our previous study of PlanetP [5]. The results show that 258 diffs that changed the content of the files led to a combined gossiping volume of 13.9MB when aggregated across all nodes in a community of 12 nodes. This network usage is negligible, giving an average of 0.46MB of network traffic per day, or .04 MB per day for each node. As each CVS commit may correspond to multiple open/modify/close edit sessions, this estimate may be conservative. Even if we assume an average of 10 edit sessions per commit, however, this would translate to only 4.6MB per day.

7 Related Work

In this section, we briefly relate Wayfinder to previous efforts spanning global namespaces, disconnected operation, semantic and P2P file systems, as well as relaxed consistency models.

Wayfinder differs from previous works building global directory structures [18, 21] in that it provides an adjustable namespace to accommodate the constraints of P2P environments. The Federated File System [25] is probably closest to ours concerning namespace construction but FFS is targeted specifically for a cluster rather than a P2P community.

The Semantic File System [8] introduced the concept of semantic directories, which was further developed in the HAC File System [9]. Wayfinder implements this abstraction in the context of P2P systems.

The Coda File System introduced the concept of hoarding for disconnected operation [14]. Seer [15] attempted to improve hoarding using semantic distance and clustering for prefetching content. To date, we have concentrated on availability as opposed to hoarding given that Wayfinder can leverage these previous works.

Many projects have recently explored P2P file systems. To our knowledge, none of these systems have considered content search, however. Further differences are as follows. The Secure Read-Only File System [7] and the Cooperative File System [6] are block-level, read-only publishing file systems. Ivy [17] is a general file system that stores data blocks in a DHT. This approach is fundamentally different from ours in that the DHT attempts to keep all data online all of the time. Wayfinder targets highly dynamic communities, however, where such a model can lead to unacceptably high data movement [3]. Pasta [16], another P2P file system, differs from Wayfinder in that its support for collaborative management of directory structures can lead to the creation of unique directory structures for individual users.

Finally, a number of projects have experimented with weak consistency models. In Bayou [20], nodes operate in a mostly disconnected mode and use a form of gossiping to exchange state. Updates are committed permanently through a primary commit server. Similarly, the OceanStore prototype Pond [22] uses replicated commit servers to accept updates. The Pangea [23] file system relies on the construction of efficient communication structures for diff propagation. Unlike these projects, Wayfinder's optimistic strategy avoids the need for commit servers but may require users to resolve conflicts. Wayfinder's optimistic approach

is similar in spirit to Ficus [10]. Ficus, however, relies on several communal two-phase algorithms to ensure consistency.

8 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 the three critical abstractions exported by Wayfinder: a unified namespace that merges multiple directory views, content addressing via semantic directories, and a probabilistic non-uniform availability model. We have also described how to build a robust file system that realizes these abstractions using a combination of weakly consistent replication through gossiping and caching of soft state using a lightweight unreliable DHT. Finally, we have shown that a prototype gives reasonable performance.

We are in the process of evaluating our implementation of ACR and hope to have more availability results if this paper is accepted. In the future, we plan to unify ACR with hoarding to support the user-centric availability metric discussed earlier. Finally, we would like to evaluate Wayfinder's actual impact on everyday data management tasks. In particular, we plan to use Wayfinder in our laboratory to manage our shared lab paper repository and blogs. (A couple of the authors were already editing this paper on Wayfinder for a short time.)

Acknowledgments

We thank Kien Le for developing the current PlanetP/Wayfinder indexer and Chunling Hu for early exploration of the security model.

References

[1] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2003.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970.

[3] 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.

[4] F. M. Cuenca-Acuna and T. D. Nguyen. Text-Based Content Search and Retrieval in ad hoc P2P Communities. In *International Workshop on Peer-to-Peer Computing*, 2002.

[5] 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 *Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.

[6] 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)*, 2001.

[7] 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)*, October 2000.

[8] 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)*, 1991.

[9] 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)*. USENIX, 1999.

[10] 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*, 1990.

[11] D. Harman. Overview of the first TREC conference. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1993.

[12] 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*, 6(1), 1988.

[13] Java nfs server. http://members.aol.com/_ht_a/ markmitche11/jnfsd.htm, October 2002.

[14] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, 1991.

[15] G. H. Kuenning and G. J. Popek. Automated hoarding for mobile computers. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, pages 264–275, Oct. 1997.

[16] T. D. Moreton, I. A. Pratt, and T. L. Harris. Storage, mutability and naming in pasta. In *International Workshop on Peer-to-Peer Computing*, 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)*, 2002.

[18] B. C. Neuman. The prospero file system: A global file system based on the virtual system model. *Computing Systems*, 5(4), 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 *7th ACM SIGOPS European Workshop*, Connemara, Ireland, 1996.

[21] H. C. Rao and L. L. Peterson. Accessing files in an internet: The jade file system. *Software Engineering*, 19(6), 1993.

[22] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The oceanstore prototype, 2003.

[23] Y. Saito and C. Karamanolis. Pangaea: a symbiotic wide-area file system. In *Proceedings of the ACM SIGOPS European Workshop*, 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] 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, 2001.