Don't Give Up on Distributed File Systems

Jeremy Stribling, Emil Sit, M. Frans Kaashoek, Jinyang Li[†] and Robert Morris
MIT CSAIL [†]*New York University*

Abstract

Wide-area distributed applications often reinvent the wheel for their storage needs, each incorporating its own special-purpose storage manager to cope with distribution, intermittent failures, limited bandwidth, and high latencies. This paper argues that a distributed file system could provide a reusable solution to these problems by coupling a standard interface with a design suited to wide-area distribution. For concreteness, this paper presents such a file system, called WheelFS, which allows applications to control consistency through the use of *semantic cues*, and minimizes communication costs by adhering to the slogan *read globally, write locally*. WheelFS could simplify distributed experiments, CDNs, and Grid applications.

1 Introduction

Distributed applications commonly require the sharing of storage between their components. Today's systems rely on specialized storage and data transfer mechanisms: some cooperative web caches use DHT techniques to replicate data, PlanetLab experiments use `scp` to centrally collect results, and Grid applications have customized file systems that position data close to computation. This paper argues that it is possible and desirable to build a distributed file system that can serve as the storage component for many wide-area applications.

Designers of distributed applications could simplify their lives by delegating many storage concerns to a distributed file system. A single filesystem name space visible from all nodes would save each application from having to provide its own naming and data lookup scheme. Similarly useful would be infrastructure support for fault-tolerance, efficient wide-area data location and transfer, and control over consistency and failure semantics.

Are there storage requirements that are common across many distributed applications? Can these requirements be met by a single general-purpose file system design? We believe the answer is yes, but that existing distributed file systems, such as AFS [31], NFS [30], and SFS [22], are not sufficient. Our experience with SFS on the PlanetLab and RON testbeds suggests two reasons.

First, most popular distributed file systems stress transparency: a remote file should behave just like one on a local-disk file system. This forces the file system to implement expensive strict consistency semantics, as well as to try hard (via long timeouts) to mask temporary server failures. Many wide-area applications don't need these semantics; for example, a cooperative web cache would rather re-fetch from the origin server than wait for the file system to find the latest copy of a cached page.

Second, widely-used distributed file systems typically store entire subtrees (or the whole file system) at a single site. This property is awkward when output generated at one site is input at another; sending data through a central server creates a needless bottleneck. Centralization is also a problem for applications that are distributed to increase availability.

The need for a global file system has been articulated for PlanetLab (Section 3 of [3]) and GENI (see the distributed services document on the GENI web site [15]), and experience with Grids also suggests that such a file system would be valuable [19]. Though it operates at a cluster level, Google's GFS [16] suggests that a shared distributed storage infrastructure can benefit many different applications.

This paper presents the design of WheelFS, a global file system with convenient location-independent names that gives applications control over data placement, failure behavior, and consistency. WheelFS incorporates the following new ideas. First, it augments the POSIX interface with *semantic cues* that allow applications to select desired behavior in the presence of failures or inconsistent updates. Second, it implements a policy of *writ-*

This research was conducted as part of the IRIS project (`http://project-iris.net/`), supported by the National Science Foundation under Cooperative Agreement No. ANI-0225660.

ing locally and reading globally, so that outputs can be written quickly and inputs can be fetched directly from the copy that is nearest on the Internet. Finally, the design relaxes certain POSIX semantics in a way that we believe will harm few practical applications yet allow the system to provide good performance. It is not intended to store users' home directories, due to these weakened semantics and a focus on performance and flexibility rather than durability; instead, we intend WheelFS to function as the storage component of large-scale distributed applications, allowing users to concentrate on their application's details and semantics, rather than data storage issues.

2 Design Sketch

WheelFS uses resources on many wide-area nodes to present users with a single data store, accessible from any of the nodes. It presents the same tree of directories and files to all nodes. WheelFS exports a standard POSIX interface (so that existing programs will work), with some extensions and modifications.

2.1 Basic design

A WheelFS file system appears as a tree of named directories and files. Internally, each file and directory has a unique numeric ID. Each participating node also has a node ID. Nodes determine responsibility using consistent hashing [18]: the node whose ID most closely follows a file or directory ID is responsible for that file or directory. Every node keeps a copy of the whole set of nodes for fast consistent hashing. These lists are kept consistent on all nodes by a Paxos-replicated [21] "configuration service" to ensure that all nodes agree on the mapping of IDs to responsible nodes. The system goes through a sequence of numbered "epochs" and responsibility for each file/directory has a linear history of one responsible node per epoch.

The responsible node holds the authoritative copy of the relevant data, processes updates, and (for files) maintains a list of nodes believed to have cached copies. The next few nodes in ID space act as replicas. The responsible node for a file or directory could change if a node joins or leaves the system, and additional replicas may then need to be created.

Files are versioned. A new version only becomes visible when the writing application calls `close()`. These semantics, inspired by AFS [31], help with efficiency and consistency. A file's responsible node knows the latest version number.

WheelFS will trust the participating node administrators (for example, it will not encrypt cached or replicated data). However, each file/directory will have an access control list (ACL) specifying the users who can read and write the file and change the ACL. We expect user (or application) identities in WheelFS to span all nodes. Identity should be supplied by the larger environment (e.g. PlanetLab [6], Globus [2], or Condor [23]), since the operating system and remote invocation mechanisms need to know about identity as well.

2.2 Semantic cues and failure recovery

WheelFS allows an application to provide *explicit semantic cues* that inform WheelFS's handling of concurrent updates (consistency) and failures. Semantic cues address the problem that no fixed semantics can provide acceptable performance and correctness to all applications. We envision the set of cues shown in Table 1. The exact set, and their exact meaning, requires further research.

Some cues are permanently attached to a file or directory when an application creates it. Others apply to a specific application reference to a file or directory (a file descriptor or working directory). For compatibility, applications will specify semantic cues in path names. For example, `/wfs/rtm/.anyversion/data` refers to `/wfs/rtm/data` with cue **AnyVersion**. Multiple cues may be specified.

For each **Strict** file and directory (and even **Lax** ones after failures heal) there will be at most one responsible node that maintains consistency by serializing updates. When the node responsible for a file/directory fails, the new responsible node must find a copy of the relevant data. WheelFS uses primary/backup replication for this: the responsible node sends every update to its two successors. After a failure, the new responsible node can find the data at one of those successors; in the common case, the node will have been a successor of the failed node. If both successors are also dead, the new responsible node must by default delay operations on the file/directory until one of them revives. This is one situation where the **WriteOnce**, **AnyVersion**, and **Lax** cues can allow progress despite failures.

WriteMany: *(file, default)* New versions of this file may be created.

WriteOnce: *(file)* Only one version of this file may be created. This cue allows WheelFS to know that any cached copy of the file is up to date; it need not check with the responsible node to find the latest version number.

LatestVersion: *(file reference, default)* When opening a file, instructs WheelFS to check with the file's responsible node to ensure that it finds the latest version.

AnyVersion: *(file reference)* When opening a file, instructs WheelFS to use the first version it can find. Allows WheelFS to avoid checking with the file's responsible node to find the latest version number.

BestVersion: *(file reference)* When opening a file, instructs WheelFS to use the highest version found within the time limit specified by the **MaxTime** cue (see below).

Strict: *(file or directory, default)* Instructs WheelFS to ensure that new file versions and directory modifications occur one at a time in a serial order. However, these operations cannot proceed if the responsible node cannot be contacted.

Lax: *(file or directory)* Allows WheelFS to let multiple nodes act as a given file or directory's responsible node in certain failure cases such as network partition. The result may be that the same version number is given to two different file versions, or that directory operations proceed without seeing the most recent modifications. When the failures heal, WheelFS will resolve conflicting operations arbitrarily but deterministically. Useful when the application knows it will never try to create the same file name from two different nodes.

MaxTime=*T*: *(file or directory reference, default T is infinite)* Specifies the maximum total wall-clock time any one operation is allowed to consume. If an operation (such as `open()`) hasn't completed in time, the operation will return an error or (with **BestVersion**) the highest numbered version found.

Table 1: Semantic cues.

2.3 Write locally / Read globally

WheelFS adopts a policy of initially storing written data on or near the writing node, to get disk or LAN throughput for output, and of searching for nearby cached copies when reading. This policy effectively results in lazy as-needed transfers, and is likely to perform well for many workloads such as intermediate result files in Grid computations.

When an application creates a new file, WheelFS picks a semi-random new file ID that will cause the application's node to be the file's responsible server. WheelFS will buffer writes to the file until the application calls `close()`, at which point it will create the first version of the file locally. This local transfer will be efficient for large files, though for small files the expense may be dominated by communicating with the directory server. Creating a new version of an existing file whose responsible server is far away will not be as efficient, so applications should store output in unique new files where possible.

In order to open and read the latest version of a file, the application's node first translates the path name by iteratively looking up each path component in the relevant directory to find the next ID, and using consistent hashing to turn the ID into the IP address of the node responsible for the next directory in the path name. This process ends with the application node contacting the node responsible for the file's ID, asking it for the latest version number and a list of IP addresses of nodes believed to be caching or replicating the file. If WheelFS finds more than one copy of the latest version, it fetches different pieces of the file from different copies using a BitTorrent-inspired algorithm and caches it locally on disk to assist other nodes. This mechanism will make reading popular files efficient, as when a distributed computation first starts.

3 Example Application

WheelFS is designed to make it easier to construct and run distributed applications. As an example, this section presents a simplified application design for a cooperative web cache: a web proxy (similar to CoralCDN [14]) that serves each requested page from a cache in WheelFS when possible, and otherwise fetches the page from the origin server.

WheelFS provides a good platform for a cooperative cache system because the application can use location-independent file names to find copies of web pages stored by other nodes. We use DNS redirection [14] to forward each client's request to a random node in our system. The node's `inetd` server will start the script shown in Figure 1 upon each incoming connection. First, the script parses the incoming HTTP request to extract the URL and

```
#!/bin/sh
# extracts URL from HTTP GET request
URL=`awk '$1 ~/GET/ { print $2 }'`
# turn URL to file name (e.g. abc.com/foo/bar.html becomes 9/f/9fxxxxxx)
FILE=`echo $URL | sha1sum | sed "s/\(.\)\(.\)/\1\/\2\/\1\2/"`
FILE_RD=/wfs/cwc/.maxtime=5,bestversion/${FILE}
FILE_WR=/wfs/cwc/.writemany,lax/${FILE}
if [ -f $FILE_RD ]; then
  EXPIRE_DATE=`head -n 100 $FILE_RD | awk '$1 ~/Expires:/ {print $2}'`
  if notexpired $EXPIRE_DATE `date`; then
    cat $FILE_RD; exit
  else
    DATE=`head -n 100 $FILE_RD | grep Date: | cut -f 2- -d " "`
  fi
else
  mkdir -p `dirname $FILE_WR`
fi
wget --header="If-Modified-Since: $DATE" --save-headers -T 2 -O $FILE_WR $URL
cat $FILE_RD
```

Figure 1: A simplified implementation of a cooperative web cache atop WheelFS. We omit the details of comparing two dates using a fake command `notexpired`.

translates it to a file name in WheelFS. For example, the URL `http://abc.com/foo/bar.html` will become `/wfs/cwc/9/f/9fxxxxxx` where `9fxxxxxx` is the content hash of the URL. If the file exists, the script opens it with cue **MaxTime=5,BestVersion**, attempting to retrieve the latest version of the file when there is no failure and using any cached copy otherwise. Lastly, the script checks if the web page has expired according to the saved HTTP header. If any step fails, the script fetches the requested URL from the original web site and saves it in WheelFS.

The cooperative web cache benefits from WheelFS's relaxed consistency semantics. The application can use a cached copy as long as it has not yet expired, and can give up quickly if there are failures. Copies of the application on different nodes may sometimes write the same file name simultaneously; the **Lax** cue allows this to result in versions with the same number or files with the same name if there are failures. Since any copy of the fetched page is adequate, WheelFS's conflict resolution strategy of eventually picking one version arbitrarily is sufficient.

One reason for the script's simplicity is that it inherits one of CoralCDN's main performance-optimizing techniques from WheelFS: reading from the closest cached copy. Both CoralCDN and our application write downloaded copies of web pages locally. Our system incurs at most 3 round-trips to look up directory entries in a file's pathname while CoralCDN requires at most O($log\ n$) round-trips, though the average number of such round-trips will decrease through caching in both systems. WheelFS contacts a file's responsible node to obtain a list of candidate nodes with cached files in order to pick a closest copy to avoid swamping any single node, while CoralCDN uses a more scalable multi-hop lookup protocol to find a nearby cached page. Part of our future work is to investigate how much performance and scalability the system actually achieves in the real world as compared to CoralCDN. Figure 1 is only a prototype as opposed to a complete final system since it does not yet address other important issues like cache resource allocation and security policies [34,35]. However, it does demonstrate how WheelFS can dramatically simplify the distributed storage management aspect of such cooperative cache systems.

4 Related Work

WheelFS adopts the idea of a configuration service from Rosebud [29] and Chubby [8], distributed directories from xFS [4], Ceph [36], Farsite [1, 12], and GFS [16], on-disk caches and whole-file operations from AFS [31], consistent hashing from CFS [11], cooperative reading from Shark [5], OceanStore/Pond [28], Coblitz [25], and BitTorrent [10], trading performance versus consistency from JetFile [17] and TACT [38], replication techniques from DHTs [9], finding the closest copy from cooperative Web caches [14, 34], and proximity prediction from location services [24].

Central-server file systems that store and serve files from a single location [4,16,22,30,31,36] create an unnecessary bottleneck when there is interaction among client hosts.

Symmetric file systems increase performance and availability by storing files on many hosts, often writing files locally and fetching data directly from peers [1, 5, 11, 12, 17, 28]. Particularly close to WheelFS are JetFile and Farsite. JetFile [17] is designed for the wide area, stores data on all hosts, provides a POSIX interface, trades some consistency for performance, uses file versioning, but relies on IP multicast. Farsite [12] is designed for a high-bandwidth, low-latency network. This assumption allows Farsite to provide strong semantics and exact POSIX compatibility with good performance. Farsite handles Byzantine failures, while WheelFS does not.

Storage systems for computational Grids [2, 7, 13, 20, 26, 27, 32, 33, 37] federate single-site storage servers, with a job's scheduler moving data among sites to keep it close to the computations. IBP [26] and GridFTP [2] are low-level mechanisms for staging files, but don't provide the convenience of a file system. LegionFS [37] provides a file system interface built on a object store.

Compared to previous files systems, WheelFS's main technical advantages are its semantic cues, inexpensive (though weak) default consistency, write-local read-global data movement, and BitTorrent-style cooperative reading, These differences reflect that WheelFS is designed as a reusable component for large-scale distributed applications.

5 Discussion

This paper has argued that a file system designed for distributed applications and experiments running across the Internet would simplify the systems' infrastructure, and presented the design of such a file system, WheelFS. The design has many open questions that will need to be explored. For example, WheelFS cannot easily handle cross-directory renames atomically; it is not clear if this would be significant problem for real applications. Nodes may need a way to offload storage burden if they are low on disk space. Files may become unreferenced if all replicas of a directory are lost due to permanent failures; perhaps a repair program will be needed. While the system is not intended for long-term durability, some attention may be needed to ensure it is durable enough for typical distributed applications. The semantic cues and default behaviors listed above are a starting point; experience will guide us towards the most useful design.

Acknowledgments

Thanks to Jeremy Kepner, Michael Freedman, and the IPTPS reviewers for their useful feedback.

References

[1] Adya, A., Bolosky, W. J., Castro, M., Cermak, G., Chaiken, R., Douceur, J. R., Howell, J., Lorch, J. R., Theimer, M., and Wattenhofer, R. P. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th OSDI* (Dec. 2002).

[2] Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., and Foster, I. The Globus striped GridFTP framework and server. In *Proceedings of the 2005 Super Computing* (Nov. 2005).

[3] Anderson, T., and Roscoe, T. Learning from PlanetLab. In *Proceedings of the 3rd WORLDS* (Nov. 2006).

[4] Anderson, T. E., Dahlin, M. D., Neefe, J. M., Patterson, D. A., Roselli, D. S., and Wang, R. Y. Serverless network file systems. In *Proceedings of the 15th SOSP* (Dec. 1995).

[5] Annapureddy, S., Freedman, M. J., and Mazières, D. Shark: Scaling file servers via cooperative caching. In *Proceedings of the 2nd NSDI* (May 2005).

[6] Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., and Wawrzoniak, M. Operating systems support for planetary-scale network services. In *Proceedings of the 1st NSDI* (Mar. 2004).

[7] Bent, J., Venkataramani, V., LeRoy, N., Roy, A., Stanley, J., Arpaci-Dusseau, A., Arpaci-Dusseau, R., , and Livny, M. NeST—a grid enabled storage appliance. *Grid Resource Management* (Sept. 2003).

[8] Burrows, M. The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th OSDI* (Nov. 2006).

[9] Chun, B.-G., Dabek, F., Haeberlen, A., Sit, E., Weatherspoon, H., Kaashoek, M. F., Kubiatowicz, J., and Morris, R. Efficient replica maintenance

for distributed storage systems. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation* (May 2006).

[10] COHEN, B. Incentives build robustness in BitTorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems* (June 2003).

[11] DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I. Wide-area cooperative storage with CFS. In *Proceedings of the 18th SOSP* (Oct. 2001).

[12] DOUCEUR, J. R., AND HOWELL, J. Distributed directory service in the Farsite file system. In *Proceedings of the 7th OSDI* (Nov. 2006).

[13] FOSTER, I., AND KESSELMAN, C. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing 11*, 2 (1997), 115–128.

[14] FREEDMAN, M. J., FREUDENTHAL, E., AND MAZIÈRES, D. Democratizing content publication with Coral. In *Proceedings of the 1st NSDI* (Mar. 2004).

[15] GENI: Global environment for network innovations. `http://www.geni.net`.

[16] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The Google file system. In *Proceedings of the 19th SOSP* (Dec. 2003).

[17] GRÖNVALL, B., WESTERLUND, A., AND PINK, S. The design of a multicast-based distributed file system. In *Proceedings of the 3rd OSDI* (Feb. 1999).

[18] KARGER, D., LEHMAN, E., LEIGHTON, F., LEVINE, M., LEWIN, D., AND PANIGRAHY, R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th Symposium on Theory of Computing* (May 1997).

[19] KOLA, G., KOSAR, T., FREY, J., LIVNY, M., BRUNNER, R. J., AND REMIJAN, M. DISC: A system for distributed data intensive scientific computing. In *Proceedings of the 1st WORLDS* (Dec. 2004).

[20] KOSAR, T., AND LIVNY, M. Stork: Making data placement a first class citizen in the grid. In *Proceedings of the 24th ICDCS* (Mar. 2004).

[21] LAMPORT, L. The part-time parliament. *ACM Transactions on Computer Systems 16*, 2 (1998), 133–169.

[22] MAZIÈRES, D., KAMINSKY, M., KAASHOEK, M. F., AND WITCHEL, E. Separating key management from file system security. In *Proceedings of the 17th SOSP* (Dec. 1999).

[23] M.J.LITZKOW, LIVNY, M., AND M.W.MUTKA. Condor: a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems* (June 1988).

[24] NG, T. S. E., AND ZHANG, H. Predicting Internet network distance with coordinates-based approaches. In *Proceedings of the 2002 Infocom* (June 2002).

[25] PARK, K., AND PAI, V. S. Scale and performance in the CoBlitz large-file distribution service. In *Proceedings of the 3rd NSDI* (May 2006).

[26] PLANK, J. S., BECK, M., ELWASIF, W., MOORE, T., SWANY, M., AND WOLSKI, R. The Internet Backplane Protocol: Storage in the network. In *Proceedings of the Network Storage Symposium* (Oct. 1999).

[27] RAJASEKAR, A., WAN, M., MOORE, R., SCHROEDER, W., KREMENEK, G., JAGATHEESAN, A., COWART, C., ZHU, B., CHEN, S.-Y., AND OLSCHANOWSKY, R. Storage resource broker—managing distributed data in a grid. *Computer Society of India Journal, Special Issue on SAN 33*, 4 (Oct. 2003).

[28] RHEA, S., EATON, P., GEELS, D., WEATHERSPOON, H., ZHAO, B., AND KUBIATOWICZ, J. Pond: The OceanStore prototype. In *Proceedings of the 2nd FAST* (Mar. 2003).

[29] RODRIGUES, R., AND LISKOV, B. Rosebud: A scalable byzantine-fault-tolerant storage architecture. *MIT LCS Technical Report TR/932* (Dec. 2003).

[30] SANDBERG, R., GOLDBERG, D., KLEIMAN, S., WALSH, D., AND LYON, B. Design and implementation of the Sun network filesystem. In *Proceedings of the Summer 1985 USENIX* (June 1985).

[31] SATYANARAYANAN, M., HOWARD, J. H., NICHOLS, D. A., SIDEBOTHAM, R. N., SPECTOR, A. Z., AND WEST, M. J. The ITC distributed file system: Principles and design. In *Proceedings of the 10th SOSP* (Dec. 1985).

[32] TATEBE, O., SODA, N., MORITA, Y., MATSUOKA, S., AND SEKIGUCHI, S. Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing. In *Proceedings of the 2004 Computing in High Energy and Nuclear Physics* (Sept. 2004).

[33] THAIN, D., BASNEY, J., SON, S.-C., AND LIVNY, M. The Kangaroo approach to data movement on the grid. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)* (Aug. 2001).

[34] WANG, L., PAI, V., AND PETERSON, L. The effectiveness of request redirecion on CDN robustness. In *Proceedings of the 5th OSDI* (Dec. 2002).

[35] WANG, L., PARK, K., PANG, R., PAI, V. S., AND PETERSON, L. Reliability and security in the CoDeeN content distribution network. In *Proceedings of the USENIX 2004 Annual Technical Conference* (Boston, MA, June 2004).

[36] WEIL, S. A., BRANDT, S. A., MILLER, E. L., LONG, D. D. E., AND MALTZAHN, C. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th OSDI* (Nov. 2006).

[37] WHITE, B. S., WALKER, M., HUMPHREY, M., AND GRIMSHAW, A. S. LegionFS: A secure and scalable file system supporting cross-domain high-performance applications. In *Proceedings of the 2001 Super Computing* (Nov. 2001).

[38] YU, H., AND VAHDAT, A. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM TOCS 20*, 3 (Aug. 2002), 239–282.