Computer Sciences Department

Stop All File Systems Research

Andrea Arpaci-Dusseau
Remzi Arpaci-Dusseau

Technical Report #1466

January 2003

UNIVERSITY OF
WISCONSIN
MADISON

Stop All File Systems Research

Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau

Computer Sciences Department
University of Wisconsin, Madison

Abstract

We advocate that the OS community should **stop all file systems research**, *because file systems are fundamentally the wrong place in the storage hierarchy to innovate. Three factors combine to limit the impact of file systems research: the constraints of legacy, the diversity of implementations, and a lack of information about storage-system internals. We instead suggest that researchers focus their innovations on storage systems, namely within disk or RAID subsystems. To enable a full range of functionality within the storage system, we posit that the disk system must be "semantically smart"; that is, it must understand the on-disk structures and recognize the on-line operations of the file system above. We discuss the concepts underlying semantically-smart disk systems, present a taxonomy of the different axes of semantic knowledge, and discuss the important remaining research challenges.*

1 Background

File systems have long been a topic of study within the realm of systems research. Perhaps the modern era of file systems research began with the introduction of the UNIX operating system [22], an operating system that is centered around the concept of a file and a hierarchical naming system [4]. Soon after its introduction, researchers realized that a serious contribution could be made to the field by replacing the standard UNIX file system with a new and improved one. An early example of such innovation is the Berkeley Fast File System (FFS) [18]; later, more sweeping changes were proposed [23, 29].

For a more quantitative study of the history of file systems research, consider publications in popular systems conferences such as the Symposium on Operating Systems Principles (SOSP), Operating Systems Design and Implementation (OSDI), and the USENIX Technical Conferences. Figure 1 presents the percentage of file systems papers published in each of those conferences since their inception. From the figure, we can observe that there has been a steady stream of file systems research. SOSP typically has one to two tracks devoted to the topic, peaking in 1991, when almost half of the accepted papers were about file systems; USENIX and OSDI follow similar trends. Although the fever pitch surrounding file systems seems to have slowed, we can still expect a reasonable number of publications on the topic.

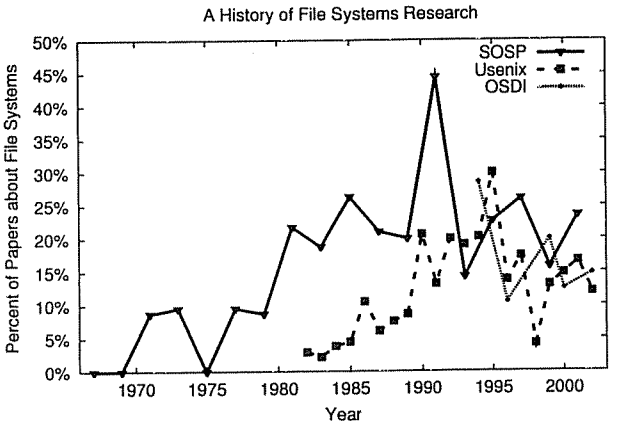

Figure 1: **File Systems Research.** *The figure presents the percent of all papers published in SOSP, USENIX, and OSDI, that were "file systems" papers. We define a file systems paper as one that is primarily concerned with the design, implementation, or measurement of some aspect of a file system. Note that until 1994, USENIX ran two conferences per year, the Summer and Winter conferences; in 1995, these were merged into the single Annual Technical Conference.*

2 The Pending File Systems Crisis: Lots of Research, Little to Show

Unfortunately, while research in file systems has proliferated, the transfer of this wealth of research ideas into the commercial world has been quite limited. As a simple example, let us examine the evolution of UNIX file systems. Introduced originally in the early 1970's [22], the UNIX file system received a substantial change in the early 1980's due to the work on the Fast File System at Berkeley [18]. The major innovation put into place was

the *cylinder group*, which allowed the file system to take locality into account during layout decisions. Many modern UNIX file systems still utilize the same on-disk structures as FFS, now nearly twenty years old. For example, the Linux ext2 file system [32], the many BSD FFS variants [6], Solaris UFS [21], and HP-UX HFS [14] all have the same basic organization as FFS.

Another advance in file systems has come in the form of journaling; many modern file systems support write-ahead logging in order to reduce crash recovery time. However, even this advance points back fifteen years, specifically to Hagmann's work on Cedar [13].

Why are file systems so slow to change? Three fundamental reasons combine to limit the deployment of new ideas within file systems. First, file systems suffer from a legacy effect. Once data is stored in a particular on-disk format, migrating to a new (and perhaps improved) organization is problematic; all older file system data must be transformed into the newer format, or the file-system code must explicitly account for multiple data formats. Hence, as new innovations are developed, they must be retrofitted into the existing on-disk structures [6]. Those file systems that require more radical restructuring may find mainstream acceptance more difficult [23].

Second, there is the problem of file system diversity. As there are many extant file systems, a new idea that is implemented within one file system leaves all other file systems unchanged; to have widespread impact, the idea must be adopted and supported by many diverse interests. Such a transfer of ideas may even be challenging among systems that support similar internal file-system interfaces, such as the the ubiquitous vnode/VFS layer [24].

Third, file systems have a limited understanding of the internals of the storage system. Built upon standardized interfaces such as SCSI or IDE, file systems typically do not understand the low-level details of the storage system. A block-based interface such as SCSI abstracts many of the details of storage management from the client, thus inhibiting many opportunities for improvements in functionality and performance [15, 30]. Although there is some desire to evolve the interface between file systems and storage [10], the reality is that current interfaces will likely survive much longer than anticipated. As Bill Joy once said, "systems may come and go, but protocols live forever."

Thus, file systems technology is fundamentally difficult to transfer to the "real world", due to the constraints of legacy, the diversity of implementations, and the inherent lack of storage information available at the file-system level. By its very nature, file systems research is likely to have little impact. We conclude that the community should immediately **stop all file systems research.**

3 A Caveat

In fairness, not all of the file systems research has been on *local* file systems, *i.e.*, file systems that sit on top of a disk or RAID within a single machine. Indeed, much of the literature concentrates upon *distributed* file systems, such as the classic USENIX paper on NFS [26]. Approximately half of the papers shown in Figure 1 focus on the topic of distributed file systems.

Work on distributed file systems continues today in the form of peer-to-peer systems, such as Ivy [19], FARSITE [1], and PAST [25]. In this paper, we take the position that research on *local* file systems should stop. In contrast, we believe that distributed file systems research should continue, due to the ever-increasing presence and importance of remote and possibly mobile data access.

4 A Solution: Storage Systems

Let us assume we all agree: stop all file systems research. However, as the data demands of applications increase, and both users and companies increasingly rely upon prompt and reliable access to their data, data management is clearly an area that should not be ignored. New problems are on the horizon as well, as users demand low-cost, reliable, available, and even easy-to-manage "no futz" systems [27].

How can researchers address these important challenges while avoiding the pitfalls of file systems research? By innovating at a lower-level in the storage hierarchy, within a disk array [20] or perhaps even within the disk itself [8, 31]. We will refer to this layer in this hierarchy as the "storage system".

Innovation within the storage system avoids many of the problems found within the file system. First, there is no legacy effect; new storage systems are plugged in and can be used immediately. Second, there is no worry of diversity; RAIDs can be deployed easily underneath all file systems that are built upon a standard abstraction such as SCSI [34]. Third, within a disk or a RAID, one has access to all of the low-level details of the storage system, enabling many optimizations that would be difficult if not impossible to implement within the file system proper.

Storage systems also offer a number of other advantages over file systems. For example, modern RAID systems from companies such as EMC consist of large numbers of processors and copious amounts of memory, *e.g.*, the top of the line EMC Symmetrix consists of up to 80 processors and 64 GB of memory [7]. Such a RAID system has the hardware resources to implement many different and interesting optimizations. Perhaps more importantly, industry prefers to sell hardware over software; sometimes referred to as the "metal box/cardboard box"

issue, many companies have found that it is simpler to sell a hardware product than it is to sell packaged software [12]. As some evidence of this, consider the success of storage products from companies such as Network Appliance and EMC, and contrast this to the relative dearth of file systems one can purchase. Even Google prefers to market a Google Search Appliance instead of packaging their software for general use [11].

Not surprisingly, researchers have not solely focused upon file systems; they also have invested a large amount of effort into building smarter disks and RAIDs [3, 8, 17, 28, 33, 34, 36]. This sampling of references only scratches the surface of storage-systems research – as a crude but indicative measure, a citation search for "RAID" on CiteSeer [2] returns 1670 entries.

5 And Now The Bad News: Limited Innovation

Given the wealth of storage-systems research, one must wonder why we should be concerned; even if file systems research continues, surely the researchers in storage systems will tackle the difficult problems that arise and solve them. However, in performing research in the context of storage systems, a new problem arises: only a small set of limited innovations are realizable.

In a traditional system, the file system is the entity that has all high-level knowledge. The file system implements basic abstractions such as files, directories, and a hierarchical namespace. It also manages data layout, using on-disk structures such as bitmaps and superblocks to track file-system state. The file system also typically implements some form of consistency management, likely in the form of journaling, to safely move the disk from one consistent state to the next as files are created and deleted.

The storage system, in contrast, has very little knowledge, due to the narrow interface between file systems and storage [5, 9]. All that is seen at this lower level in the hierarchy is a series of block read and block write requests; no meaning is attached to each operation.

Unfortunately, this lack of "semantic" knowledge starkly limits the types of innovations that can be implemented within the storage system. Not surprisingly, all previous research has thus been limited to optimizations that are oblivious to the nature and meaning of file system traffic. Without semantic knowledge, the scope of new functionality that can be implemented within the storage system is severely curtailed.

6 A Better Solution: Semantically-Smart Disks

To enable a wider range of "file-system like" innovations within storage systems, while retaining the same narrow interface between file systems and storage, we advocate research in the area of semantically-smart disk systems. A semantically-smart disk system (SDS) has detailed knowledge of how it is used by the file system above, including an understanding of the on-disk structures and the on-line behavior of the file system. The SDS can then exploit this knowledge to improve performance, enhance functionality, or even to increase reliability.

With semantic knowledge, storage system researchers are free to innovate in a "best of both worlds" environment, for the following reasons. The systems they develop have access to high-level semantic knowledge typically isolated within the file system. In addition, these systems can exploit the raw computing and memory resources and low-level information that is often available within a modern storage system.

Our early experience with semantically-smart disks is described in [30]. Therein, we present a tool, EOF, that assists the SDS in obtaining file system on-disk layout information, and discuss the challenges in inferring the on-line behavior of the file system.

We also present four case studies to demonstrate the potential of semantically-smart disks, including on-disk caching schemes that utilize file system knowledge to perform better second-level caching, track-aligned file placement, secure-deletion of files via repeated over-write, and journaling within the disk itself. Each study demonstrates that "file-system like" optimizations can be developed within the disk system.

However, many general and important questions are left unanswered. How much knowledge is required to implement various types of functionality? Can the information that the SDS has be wrong (occasionally), or must it be perfect? How should the system as a whole be designed to enable more effective implementations? To answer these questions, we need to better understand the different degrees of freedom an SDS designer has.

7 Axes of Semantic Knowledge

We now present a taxonomy of semantic knowledge. Such a taxonomy should be of use in the design of a semantic-disk system, in deciding what types of semantic information are required, given the particular functionality that a researcher or company wishes to implement.

Similar in spirit to the different levels of RAID systems [20], we will use the taxonomy to distinguish different types of semantically-smart systems that could be de-

veloped. However, instead of a linear ordering, we believe that there are three main axes which must be considered: extent of knowledge, certainty of knowledge, and the level of expected assistance from the file system above.

7.1 Extent of knowledge

The first axis is the extent of file system knowledge present within the semantically-smart disk. At one extreme, the disk has no file system knowledge, which is the situation in most traditional systems. At the other extreme, the disk has full knowledge about file system structures and behavior.

Of course, there are many interesting knowledge-levels between the two extremes. For example, a system that needs free space on disk in order to store additional data may only need to understand file system bitmaps. A company developing a disk that performs intelligent file-based prefetching might only require knowledge of inodes and their pointers in their product. Finally, a more sophisticated semantically-smart disk would know about the current state of the file system cache, to implement a more effective exclusive second-level caching scheme [35].

7.2 Certainty of knowledge

Orthogonal to the extent of the knowledge is the semantically-smart disk's certainty of the knowledge that it possesses. At one end of the spectrum, the disk does not trust any file system knowledge it has; this might occur before the disk has configured itself for the first time. At the other end of the spectrum, the disk is absolutely certain of its knowledge, *i.e.*, it can prove to itself that what it thinks is true is indeed true. Achieving this extreme certainty will likely require a careful verification procedure.

Along this continuum, the points between the extremes all bear some similarity, in that they represent partial certainty. If the knowledge turns out to be wrong (*i.e.*, it is a "hint" [16]), correctness should not be endangered, although performance might suffer. For example, a disk could assume that certain blocks are indirect blocks and upon access, decide to prefetch the blocks to which the pointers within the indirect block point. If the disk is incorrect about the type of the indirect block (*e.g.*, it is a data block), all that is wasted is disk bandwidth. This type of optimization might be particularly well-suited in a FireWire disk product, as the internal bandwidth of modern drives currently outstrips the maximum FireWire transfer speeds of roughly 80 MB/s.

Certainty may also have a time-variant aspect. For example, when a data block is written out, the disk may not be able to infer whether it is a file data, directory, or indirect block. However, when its inode is written to disk, the disk will then be able to draw the proper conclusion.

7.3 Assistance from above

Finally, we can categorize a semantically-smart disk system by the level of assistance it expects or requires from the file system above. The completely independent SDS needs nothing from the file system beyond its normal behavior. The completely dependent SDS relies solely on the file system for all of its information.

There are many points in between the two extremes. For example, file systems such as the original FFS immediately reflect all meta-data operations through to the storage system; this type of information can be useful within an SDS, as meta-data updates are indicative of higher-level file system activity [30]. Some modern file systems can behave in this manner without modification, *e.g.*, by mounting Linux ext2 in synchronous mode. More assistance could be given as well; for example, the file system could pass explicit information to the disk about different types of file-system level knowledge. Of course, for this to be realized, an explicit communication channel must be established between the file system and the disk, likely in the form of a log file in which the file system records information it wishes the disk to observe.

Note that the type of information passed from the file system from above can be either static or dynamic. For example, the file system could inform the disk about the structure of an inode (*i.e.*, static). The file system could also inform the SDS upon every data block write whether the data block was a file data, directory, or indirect block (*i.e.*, dynamic).

8 Discussion

With a taxonomy in place, we can apply it in order to better understand previous work, and to point us towards the remaining research challenges. We consider our initial work on semantically-smart disk systems [30].

We make the following observations in the context of the taxonomy. First, all of the case studies generally require a great deal of knowledge about file system structures and behaviors. In the extreme case, the journaling semantically-smart disk must be able to understand all on-disk structures, including many fields of the inodes, and observe high-level operations such as file creations and deletions. Second, most of the case studies require information to be correct, and there is little understanding of what happens if information about the file system is incorrect. One small exception to this arises in the caching study, which can tolerate a late classification of an indirect block. Finally, our previous work assumes that file systems provided little or no assistance to the disk, although both the secure-deleting and journaling disks required a synchronous mount of ext2.

When placed into the taxonomy, future research questions come into clear focus. How can we minimize the amount of information required by a particular SDS? How sensitive is a particular SDS to the correctness of its information? What kinds of functionality require what kinds of knowledge, both in terms of the amount of knowledge as well as its certainty? Finally, perhaps as the exception that proves the rule, how much more effective can an SDS become given assistance from the file system?

9 Conclusions

Stop all file systems research – but continue research in storage, in the context of semantically-smart disk systems. A semantically-smart disk represents a "best of both worlds" environment; by developing functionality within storage system, researchers can avoid the problems that plague the dissemination of file systems research while opening up new avenues for their innovations.

Acknowledgements

The structure of this paper is an homage to Patterson, Gibson, and Katz's paper on RAID [20].

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 *OSDI '02*, Dec. 2002.

[2] K. Bollacker, S. Lawrence, and C. L. Giles. A System for Automatic Personalized Tracking of Scientific Literature on the Web. In *Digital Libraries 99 - The Fourth ACM Conference on Digital Libraries*, pages 105–113, New York, 1999. ACM Press.

[3] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[4] R. Daley and J. B. Dennis. Virtual memory, processes, and sharing in Multics. *Communications of the ACM*, 11:306–12, 1968.

[5] T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Bridging the Information Gap in Storage Protocol Stacks. In *USENIX '02*, Monterey, CA, June 2002.

[6] I. Dowse and D. Malone. Recent Filesystem Optimisations on FreeBSD. In *Proceedings of the USENIX Annual Technical Conference (FREENIX Track)*, Monterey, California, June 2002.

[7] EMC Corporation. Symmetrix Enterprise Information Storage Systems. http://www.emc.com, 2002.

[8] R. M. English and A. A. Stepanov. Loge: A Self-Organizing Disk Controller. In *Proceedings of the USENIX Winter 1992 Technical Conference*, pages 237–252, San Francisco, CA, January 1992.

[9] G. R. Ganger. Blurring the Line Between Oses and Storage Devices. Technical Report CMU-CS-01-166, Carnegie Mellon University, December 2001.

[10] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A Cost-Effective, High-Bandwidth Storage Architecture. In *ASPLOS VIII*, October 1998.

[11] Google.com. http://www.google.com/appliance/, 2003.

[12] J. Gray. *Personal Communication*, 2002.

[13] R. Hagmann. Reimplementing the Cedar File System Using Logging and Group Commit. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, November 1987.

[14] Hewlett Packard. devrsrc1.external.hp.com/STKLI/man/11iv1.5/tunefs_1m.html, 2003.

[15] M. F. Kaashoek, D. R. Engler, G. R. Ganger, H. Briceño, R. Hunt, D. Mazières, T. Pinckney, R. Grimm, J. Jannotti, and K. Mackenzie. Application Performance and Flexibility on Exokernel Systems. In *SOSP '97*, pages 52–65, 1997.

[16] B. W. Lampson. Hints for Computer System Design. *Operating Systems Review*, 17(5):33–48, October 1983.

[17] C. Lumb, J. Schindler, G. Ganger, D. Nagle, and E. Riedel. Towards Higher Disk Head Utilization: Extracting "Free" Bandwidth From Busy Disk Drives. In *OSDI '00*, October 2000.

[18] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A Fast File System for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.

[19] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: A Read/Write Peer-to-Peer File System. In *OSDI '02*, Dec. 2002.

[20] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *SIGMOD '88*, 1988.

[21] J. K. Peacock, A. Kamaraju, and S. Agrawal. Fast Consistency Checking for the Solaris File System. In *USENIX 1998 Annual Technical Conference*, New Orleans, Louisiana, June 1998.

[22] D. M. Ritchie and K. Thompson. The UNIX Time-Sharing System. *Communications of the ACM*, 17(7):365–375, 1974.

[23] M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 10(1):26–52, February 1992.

[24] D. S. H. Rosenthal. Evolving the Vnode Interface. In *Proceedings of the 1990 USENIX Summer Technical Conference*, pages 107–118, Anaheim, CA, 1990.

[25] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to- peer Storage Utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 43–56, Banff, Canada, October 2001.

[26] R. Sandberg. The Design and Implementation of the Sun Network File System. In *Proceedings of the 1985 USENIX Summer Technical Conference*, pages 119–130, Berkeley, CA, June 1985.

[27] M. Satyanarayanan. Digest of HotOS-VII. www.cs.rice.edu/Conferences/HotOS/digest/digest-html.html, March 1999.

[28] S. Savage and J. Wilkes. AFRAID — A Frequently Redundant Array of Independent Disks. In *Proceedings of the 1996 USENIX Technical Conference*, pages 27–39, San Diego, CA, January 1996.

[29] M. Seltzer, K. Bostic, M. K. McKusick, and C. Staelin. An Implementation of a Log-Structured File System for UNIX. In *USENIX Winter '93*, 1993.

[30] M. Sivathanu, V. Prabhakaran, F. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Semantically-Smart Disk Systems. In *FAST '03 (to appear)*, 2003.

[31] D. Slotnick. *Logic Per Track Devices*, volume 10, pages 291–296. Academic Press, 1970.

[32] T. Ts'o and S. Tweedie. Future Directions for the Ext2/3 Filesystem. In *Proceedings of the USENIX Annual Technical Conference (FREENIX Track)*, Monterey, California, June 2002.

[33] R. Wang, T. E. Anderson, and D. A. Patterson. Virtual Log-Based File Systems for a Programmable Disk. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*, New Orleans, LA, February 1999.

[34] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 96–108, Copper Mountain, CO, December 1995. ACM Press.

[35] T. M. Wong and J. Wilkes. My Cache or Yours? Making Storage More Exclusive. In *The Proceedings of the USENIX Annual Technical Conference (USENIX '02)*, Monterey, CA, June 2002.

[36] X. Yu, B. Gum, Y. Chen, R. Y. Wang, K. Li, A. Krishnamurthy, and T. E. Anderson. Trading Capacity for Performance in a Disk Array. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI '00)*, San Diego, CA, October 2000.