File System Design for an NFS File Server Appliance

by Dave Hitz, James Lau, and Michael Malcolm

Download:

- the Adobe Acrobat File (73Kb)
- the PostScript File (486Kb)

[TR3002]

Preface to the Reader

Network Appliance continues to evolve its technology and products at a fast pace, with significant new features and performance enhancements introduced every nine months or less over the past three years. This paper reflects the nomenclature and product characteristics in March 1995. The basic architecture, WAFL file system, and RAID-4 implementation details are fundamentally accurate and still apply to NetApp filers, even though at the time of publication *filers* were called *FAServers*. For a more current overview of NetApp's architecture please see "Multiprotocol Data Access: NFS, CIFS, and HTTP" (TR-3014).

Abstract

Network Appliance recently began shipping a new kind of network server called an NFS file server appliance, which is a dedicated server whose sole function is to provide NFS file service. The file system requirements for an NFS appliance are different from those for a general-purpose UNIX system, both because an NFS appliance must be optimized for network file access and because an appliance must be easy to use.

This paper describes WAFL™ (Write Anywhere File Layout), which is a file system designed specifically to work in an NFS appliance. The primary focus is on the algorithms and data structures that WAFL uses to implement Snapshots , which are read-only clones of the active file system. WAFL uses a copy-on-write technique to minimize the disk space that Snapshots consume. This paper also describes how WAFL uses Snapshots to eliminate the need for file system consistency checking after an unclean shutdown.

1. Introduction

An appliance is a device designed to perform a particular function. A recent trend in networking has been to provide common services using appliances instead of general- purpose computers. For instance, special-purpose routers from companies like Cisco and Bay Networks have almost entirely replaced general-purpose computers for packet routing, even though general purpose computers originally handled all routing. Other examples of network appliances include network terminal concentrators, network FAX servers, and network printers.

A new type of network appliance is the NFS file server appliance. The requirements for a file system operating in an NFS appliance are different from those for a general purpose file system: NFS access patterns are different from local access patterns, and the special-purpose nature of an appliance also affects the design. WAFL (Write Anywhere File Layout) is the file system used in all Network Appliance Corporation's file servers. WAFL was designed to meet four primary requirements:

1. It should provide fast NFS service.
2. It should support large file systems (tens of GB) that grow dynamically as disks are added.
3. It should provide high performance while supporting RAID (Redundant Array of Independent Disks).
4. It should restart quickly, even after an unclean shutdown due to power failure or system crash.

The requirement for fast NFS service is obvious, given WAFL's intended use in an NFS appliance. Support for large file systems simplifies system administration by allowing all disk space to belong to a single large partition. Large file systems make RAID desirable because the probability of disk failure increases with the number of disks. Large file systems require special techniques for fast restart because the file system consistency checks for normal UNIX file systems become unacceptably slow as file systems grow.

NFS and RAID both strain write performance: NFS because servers must store data safely before replying to NFS requests, and RAID because of the read-modify-write sequence it uses to maintain parity. This led us to use non-volatile RAM to reduce NFS response time and a write-anywhere design that allows WAFL to write to disk locations that minimize RAID's write performance penalty. The write-anywhere design enables Snapshots, which in turn eliminate the requirement for time-consuming consistency checks after power loss or system failure.

2. Introduction To Snapshots

WAFL's primary distinguishing characteristic is Snapshots, which are read-only copies of the entire file system. WAFL creates and deletes Snapshots automatically at prescheduled times, and it keeps up to 20 Snapshots on-line at once to provide easy access to old versions of files.

Snapshots use a copy-on-write technique to avoid duplicating disk blocks that are the same in a Snapshot as in the active file system. Only when blocks in the active file system are modified or removed do Snapshots containing those blocks begin to consume disk space.

Users can access Snapshots through NFS to recover files that they have accidentally changed or removed, and system administrators can use Snapshots to create backups safely from a running system. In addition, WAFL uses Snapshots internally so that it can restart quickly even after an unclean system shutdown.

2.1. User Access to Snapshots

Every directory in the file system contains a hidden sub-directory named `.snapshot` that allows users to access the contents of Snapshots over NFS. Suppose that a user has accidentally removed a file named `todo` and wants to recover it. The following example shows how to list all the versions of `todo` saved in Snapshots:

```
spike% ls -lut .snapshot/*/todo
-rw-r--r-- 1 hitz 52880 Oct 15 00:00
.snapshot/nightly.0/todo
-rw-r--r-- 1 hitz 52880 Oct 14 19:00
.snapshot/hourly.0/todo
-rw-r--r-- 1 hitz 52829 Oct 14 15:00
.snapshot/hourly.1/todo
...
-rw-r--r-- 1 hitz 55059 Oct 10 00:00
.snapshot/nightly.4/todo
-rw-r--r-- 1 hitz 55059 Oct 9 00:00
.snapshot/nightly.5/todo
```

With the **-u** option, **ls** shows todo's access time, which is set to the time when the Snapshot containing it was created. The user can recover the most recent version of todo by copying it back into the current directory:

```
spike% cp .snapshot/hourly.0/todo .
```

The .snapshot directories are "hidden" in the sense that they do not show up in directory listings. If .snapshot were visible, commands like **fin** would report many more files than expected, and commands like **rm -rf** would fail because files in Snapshots are read-only and cannot be removed.

2.2. Snapshot Administration

The FAServer has commands to let system administrators create and delete Snapshots, but it creates and deletes most Snapshots automatically. By default, the FAServer creates four hourly Snapshots at various times during the day, and a nightly Snapshot every night at midnight. It keeps the hourly Snapshots for two days, and the nightly Snapshots for a week. It also creates a weekly Snapshot at midnight on Sunday, which it keeps for two weeks.

For file systems that change quickly, this schedule may consume too much disk space, and Snapshots may need to be deleted sooner. A Snapshots is useful even if it is kept for just a few hours, because users usually notice immediately when they have removed an important file. For file systems that change slowly, it may make sense to keep Snapshots on-line for longer. In typical environments, keeping Snapshots for one week consumes 10 to 20 percent of disk space.

3. WAFL Implementation

3.1. Overview

WAFL is a UNIX compatible file system optimized for network file access. In many ways WAFL is similar to other UNIX file systems such as the Berkeley Fast File System (FFS) and TransArc's Episode file system. WAFL is a block-based file system that uses inodes to describe files. It uses 4 KB blocks with no fragments.

Each WAFL inode contains 16 block pointers to indicate which blocks belong to the file. Unlike FFS, all the block pointers in a WAFL inode refer to blocks at the same level. Thus, inodes for files smaller than 64 KB use the 16 block pointers to point to data blocks. Inodes for files smaller than 64 MB point to indirect blocks which point to actual file data. Inodes for larger files point to doubly indirect blocks. For very small files, data is stored in the inode itself in place of the block pointers.

3.2. Meta-Data Lives in Files

Like Episode, WAFL stores meta-data in files. WAFL's three meta-data files are the inode file, which contains the inodes for the file system, the block-map file, which identifies free blocks, and the inode-map file, which identifies free inodes. The term map is used instead of bit map because these files use more than one bit for each entry. The block-map file's format is described in detail below.

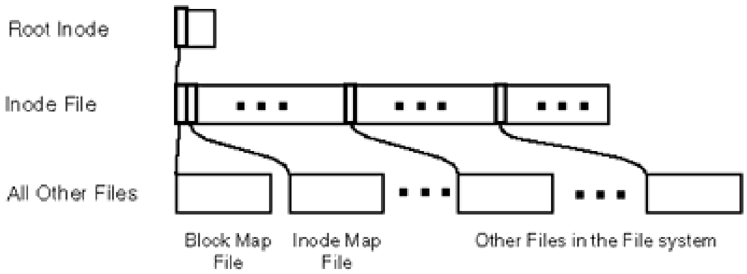

Figure 1: The WAFL file system is a tree of blocks with the root inode, which describes the inode file, at the top, and meta-data files and regular files underneath.

Keeping meta-data in files allows WAFL to write meta-data blocks anywhere on disk. This is the origin of the name WAFL, which stands for Write Anywhere File Layout. The write-anywhere design allows WAFL to operate efficiently with RAID by scheduling multiple writes to the same RAID stripe whenever possible to avoid the 4-to-1 write penalty that RAID incurs when it updates just one block in a stripe.

Keeping meta-data in files makes it easy to increase the size of the file system on the fly. When a new disk is added, the FAServer automatically increases the sizes of the meta-data files. The system administrator can increase the number of inodes in the file system manually if the default is too small. Finally, the write-anywhere design enables the copy-on-write technique used by Snapshots.

For Snapshots to work, WAFL must be able to write all new data, including meta-data, to new locations on disk, instead of overwriting the old data. If WAFL stored meta-data at fixed locations on disk, this would not be possible.

3.3. Tree of Blocks

A WAFL file system is best thought of as a tree of blocks. At the root of the tree is the root inode, as shown in Figure 1. The root inode is a special inode that describes the inode file. The inode file contains the inodes that describe the rest of the files in the file system, including the block-map and inode-map files. The leaves of the tree are the data blocks of all the files.

Figure 2 is a more detailed version of Figure 1. It shows that files are made up of individual blocks and that large files have additional layers of indirection between the inode and the actual data blocks. In order for WAFL to boot, it must be able to find the root of this tree, so the one exception to WAFL's write-anywhere rule is that the block containing the root inode must live at a fixed location on disk where WAFL can find it.

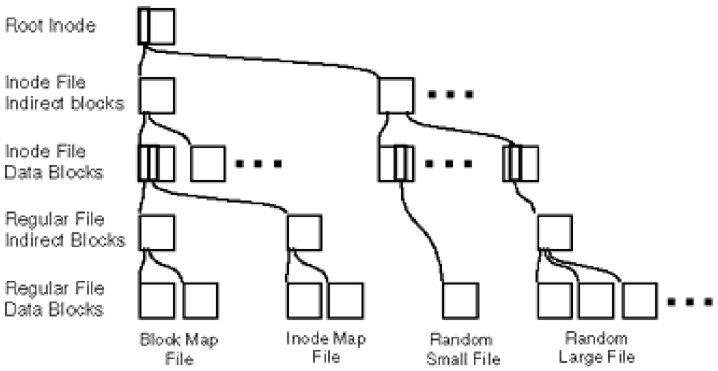

Figure 2: A more detailed view of WAFL's tree of blocks.

3.4. Snapshots

Understanding that the WAFL file system is a tree of blocks rooted by the root inode is the key to understanding Snapshots. To create a virtual copy of this tree of blocks, WAFL simply duplicates the root inode. Figure 3 shows how this works.

Figure 3(a) is a simplified diagram of the file system in Figure 2 that leaves out internal nodes in the tree, such as inodes and indirect blocks.

Figure 3(b) shows how WAFL creates a new Snapshot by making a duplicate copy of the root inode. This duplicate inode becomes the root of a tree of blocks representing the Snapshot, just as the root inode represents the active file system. When the Snapshot inode is created, it points to exactly the same disk blocks as the root inode, so a brand new Snapshot consumes no disk space except for the Snapshot inode itself.

Figure 3(c) shows what happens when a user modifies data block D. WAFL writes the new data to block D' on disk, and changes the active file system to point to the new block. The Snapshot still references the original block D which is unmodified on disk. Over time, as files in the active file system are modified or deleted, the Snapshot references more and more blocks that are no longer used in the active file system. The rate at which files change determines how long Snapshots can be kept on line before they consume an unacceptable amount of disk space.

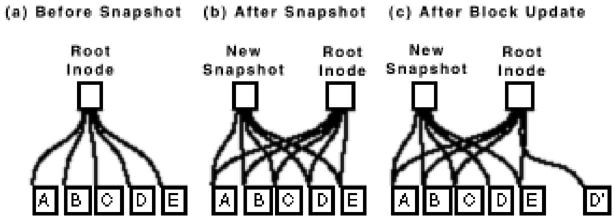

Figure 3: WAFL creates a Snapshot by duplicating the root inode that describes the inode file. WAFL avoids changing blocks in a Snapshot by writing new data to new locations on disk.

It is interesting to compare WAFL's Snapshots with Episode's fileset clones. Instead of duplicating the root inode, Episode creates a clone by copying the entire inode file. This generates considerable disk I/O and consumes a lot of disk space. For instance, a 10 GB file system with one inode for every 4 KB of disk space would have 320 MB of inodes. In such a file system, creating a Snapshot by duplicating the inodes would generate 320 MB of disk I/O and consume 320 MB of disk space. Creating 10 such Snapshots would consume almost one-third of the file system's space even before any data blocks were modified.

By duplicating just the root inode, WAFL creates Snapshots very quickly and with very little disk I/O. Snapshot performance is important because WAFL creates a Snapshot every few seconds to allow quick recovery after unclean system shutdowns.

Figure 4 shows the transition from Figure 3(b) to 3(c) in more detail. When a disk block is modified, and its contents are written to a new location, the block's parent must be modified to reflect the new location. The parent's parent, in turn, must also be written to a new location, and so on up to the root of the tree.

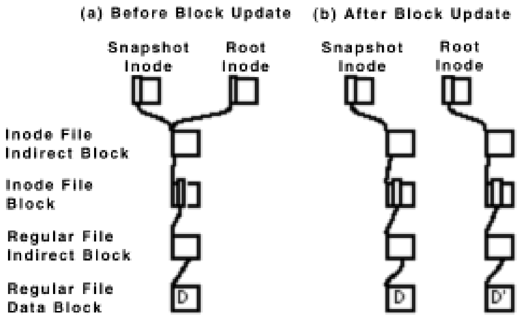

Figure 4: To write a block to a new location, the pointers in the block's ancestors must be updated, which requires them to be written to new locations as well.

WAFL would be very inefficient if it wrote this many blocks for each NFS write request. Instead, WAFL gathers up many hundreds of NFS requests before scheduling a write episode. During a write episode, WAFL allocates disk space for all the dirty data in the cache and schedules the required disk I/O. As a result, commonly modified blocks, such as indirect blocks and blocks in the inode file, are written only once per write episode instead of once per NFS request.

3.5. File System Consistency and Non-Volatile RAM

WAFL avoids the need for file system consistency checking after an unclean shutdown by creating a special Snapshot called a consistency point every few seconds. Unlike other Snapshots, a consistency point has no name, and it is not accessible through NFS. Like all Snapshots, a consistency point is a completely self consistent image of the entire file system. When WAFL restarts, it simply reverts to the most recent consistency point. This allows a FAServer to reboot in about a minute even with 20 GB or more of data in its single partition.

Between consistency points, WAFL does write data to disk, but it writes only to blocks that are not in use, so the tree of blocks representing the most recent consistency point remains completely unchanged. WAFL processes hundreds or thousands of NFS requests between consistency points, so the on-disk image of t he file system remains the same for many seconds until WAFL writes a new consistency point, at which time the on-disk image advances atomically to a new state that reflects the changes made by the new requests. Although this technique is unusual for a UNIX file system, it is well known for databases. Even in databases it is unusual to write as many operations at one time as WAFL does in its consistency points.

WAFL uses non-volatile RAM (NVRAM) to keep a log of NFS requests it has processed since the last consistency point. (NVRAM is special memory with batteries that allow it to store data even when system power is off.) After an unclean shutdown, WAFL replays any requests in the log to prevent them from being lost. When a FAServer shuts down normally, it creates one last consistency point after suspending NFS service. Thus, on a clean shutdown the NVRAM doesn't contain any unprocessed NFS requests, and it is turned off to increase its battery life.

WAFL actually divides the NVRAM into two separate logs. When one log gets full, WAFL switches to the other log and starts writing a consistency point to store the changes from the first log safely on disk. WAFL schedules a consistency point every 10 seconds, even if the log is not full, to prevent the on-disk image of the file system from getting too far out of date.

Logging NFS requests to NVRAM has several advantages over the more common technique of using NVRAM to cache writes at the disk driver layer. Lyon and Sandberg describe the NVRAM write cache technique, which Legato's Prestoserve NFS accelerator uses.

Processing an NFS request and caching the resulting disk writes generally takes much more NVRAM than simply logging the information required to replay the request. For instance, to move a file from one directory to another, the file system must update the contents and inodes of both the source and target directories. In FFS, where blocks are 8 KB each, this uses 32 KB of cache space. WAFL uses about 150 bytes to log the information needed to replay a rename operation. Rename with its factor of 200 difference in NVRAM usage is an extreme case, but even for a simple 8 KB write, caching disk blocks will consume 8 KB for the data, 8 KB for the inode update, and for large files another 8 KB for the indirect block. WAFL logs just the 8 KB of data along with about 120 bytes of header information. With a typical mix of NFS operations, WAFL can store more than 1000 operations per megabyte of NVRAM.

Using NVRAM as a cache of unwritten disk blocks turns it into an integral part of the disk subsystem. An NVRAM failure can corrupt the file system in ways that fsck cannot detect or repair. If something goes wrong with WAFL's NVRAM, WAFL may lose a few NFS requests, but the on-disk image of the file system remains completely self consistent. This is important because NVRAM is reliable, but not as reliable as a RAID disk array.

A final advantage of logging NFS requests is that it improves NFS response times. To reply to an NFS request, a file system without any NVRAM must update its in-memory data structures, allocate disk space for new data, and wait for all modified data to reach disk. A file system with an NVRAM write cache does all the same steps, except that it copies modified data into NVRAM instead of waiting for the data to reach disk. WAFL can reply to an NFS request much more quickly because it need only update its in-memory data structures and log the request. It does not allocate disk space for new data or copy modified data to NVRAM.

3.6. Write Allocation

Write performance is especially important for network file servers. Ousterhout observed that as read caches get larger at both the client and server, writes begin to dominate the I/O subsystem [Ousterhout89]. This effect is especially pronounced with NFS which allows very little client-side write caching. The result is that the disks on an NFS server may have 5 times as many write operations as reads.

WAFL's design was motivated largely by a desire to maximize the flexibility of its write allocation policies. This flexibility takes three forms:

(1) WAFL can write any file system block (except the one containing the root inode) to any location on disk. In FFS, meta-data, such as inodes and bit maps, is kept in fixed locations on disk. This prevents FFS from optimizing writes by, for example, putting both the data for a newly updated file and its inode right next to each other on disk. Since WAFL can write meta-data anywhere on disk, it can optimize writes more creatively.

(2) WAFL can write blocks to disk in any order. FFS writes blocks to disk in a carefully determined order so that fsck(8) can restore file system consistency after an unclean shutdown. WAFL can write blocks in any order because the on-disk image of the file system changes only when WAFL writes a consistency point. The one constraint is that WAFL must write all the blocks in a new consistency point before it writes the root inode for the consistency point.

(3) WAFL can allocate disk space for many NFS operations at once in a single write episode. FFS allocates disk space as it processes each NFS request. WAFL gathers up hundreds of NFS requests before scheduling a consistency point, at which time it allocates blocks for all requests in the consistency point at once. Deferring write allocation improves the latency of NFS operations by removing disk allocation from the processing path of the reply, and it avoids wasting time allocating space for blocks that are removed before they reach disk.

These features give WAFL extraordinary flexibility in its write allocation policies. The ability to schedule writes for many requests at once enables more intelligent allocation policies, and the fact that blocks can be written to any location and in any order allows a wide variety of strategies. It is easy to try new block allocation strategies without any change to WAFL's on-disk data structures.

The details of WAFL's write allocation policies are outside the scope of this paper. In short, WAFL improves RAID performance by writing to multiple blocks in the same stripe; WAFL reduces seek time by writing blocks to locations that are near each other on disk; and WAFL reduces head-contention when reading large files by placing sequential blocks in a file on a single disk in the RAID array. Optimizing write allocation is difficult because these goals often conflict.

4. Snapshot Data Structures And Algorithms

4.1. The Block-Map File

Most file systems keep track of free blocks using a bit map with one bit per disk block. If the bit is set, then the block is in use. This technique does not work for WAFL because many snapshots can reference a block at the same time. WAFL's block-map file contains a 32-bit entry for each 4 KB disk block. Bit 0 is set if the active file system references the block, bit 1 is set if the first Snapshot references the block, and so on. A block is in use if any of the bits in its block-map entry are set.

Figure 5 shows the life cycle of a typical block-map entry. At time t1, the block-map entry is completely clear, indicating that the block is available. At time t2, WAFL allocates the block and stores file data in it. When Snapshots are created, at times t3 and t4, WAFL copies the active file system bit into the bit indicating membership in the Snapshot. The block is deleted from the active file system at time t5. This can occur either because the file containing the block is removed, or because the contents of the block are updated and the new contents are written to a new location on disk. The block can't be reused, however, until no Snapshot references it. In Figure 5, this occurs at time t8 after both Snapshots that reference the block have been removed.

Time	Block-Map Entry	Description
t1	00000000	Block is unused
t2	00000001	Block is allocated for active FS
t3	00000011	Snapshot #1 is created
t4	00000111	Snapshot #2 is created
t5	00000110	Block is deleted from active FS
t6	00000110	Snapshot #3 is created
t7	00000100	Snapshot #1 is deleted
t8	00000000	Snapshot #2 is deleted; block is unused

bit 0: set for active file system
bit 1: set for Snapshot #1
bit 2: set for Snapshot #2
bit 3: set for Snapshot #3

Figure 5: The life cycle of a block-map file entry.

4.2. Creating a Snapshot

The challenge in writing a Snapshot to disk is to avoid locking out incoming NFS requests. The problem is that new NFS requests may need to change cached data that is part of the Snapshot and which must remain unchanged until it reaches disk. An easy way to create a Snapshot would be to suspend NFS processing, write the Snapshot, and then resume NFS processing. However, writing a Snapshot can take over a second, which is too long for an NFS server to stop responding. Remember that WAFL creates a consistency point Snapshot at least every 10 seconds, so performance is critical.

WAFL's technique for keeping Snapshot data self consistent is to mark all the dirty data in the cache as IN_SNAPSHOT. The rule during Snapshot creation is that data marked IN_SNAPSHOT must not be modified, and data not marked IN_SNAPSHOT must not be flushed to disk. NFS requests can read all file system data, and they can modify data that isn't IN_SNAPSHOT, but processing for requests that need to modify IN_SNAPSHOT data must be deferred.

To avoid locking out NFS requests, WAFL must flush IN_SNAPSHOT data as quickly as possible. To do this, WAFL performs the following steps:

(1) Allocate disk space for all files with IN_SNAPSHOT blocks. WAFL caches inode data in two places: in a special cache of in-core inodes, and in disk buffers belonging to the inode file. When it finishes write allocating a file, WAFL copies the newly updated inode information from the inode cache into the appropriate inode file disk buffer, and clears the IN_SNAPSHOT bit on the in-core inode. When this step is complete no

inodes for regular files are marked IN_SNAPSHOT, and most NFS operations can continue without blocking. Fortunately, this step can be done very quickly because it requires no disk I/O.

(2) Update the block-map file. For each block-map entry, WAFL copies the bit for the active file system to the bit for the new Snapshot.

(3) Write all IN_SNAPSHOT disk buffers in cache to their newly-allocated locations on disk. As soon as a particular buffer is flushed, WAFL restarts any NFS requests waiting to modify it.

(4) Duplicate the root inode to create an inode that represents the new Snapshot, and turn the root inode's IN_SNAPSHOT bit off. The new Snapshot inode must not reach disk until after all other blocks in the Snapshot have been written. If this rule were not followed, an unexpected system shutdown could leave the Snapshot in an inconsistent state.

Once the new Snapshot inode has been written, no more IN_SNAPSHOT data exists in cache, and any NFS requests that are still suspended can be processed. Under normal loads, WAFL performs these four steps in less than a second. Step (1) can generally be done in just a few hundredths of a second, and once WAFL completes it, very few NFS operations need to be delayed.

Deleting a Snapshot is trivial. WAFL simply zeros the root inode representing the Snapshot and clears the bit representing the Snapshot in each block-map entry.

5. Performance

It is difficult to compare WAFL's performance to other file systems directly. Since WAFL runs only in an NFS appliance, it can be benchmarked against other file systems only in the context of NFS. The best NFS benchmark available today is the SPEC SFS (System File Server) benchmark, also known as LADDIS. The name LADDIS stands for the group of companies that originally developed the benchmark: Legato, Auspex, Digital, Data General, Interphase, and Sun.

LADDIS tests NFS performance by measuring a server's response time at various throughput levels. Servers typically handle requests most quickly at low load levels; as the load increases, so does the response time. Figure 6 compares the FAServer's LADDIS performance with that of other well-known NFS servers.

Using a system level benchmark, such as LADDIS, to compare file system performance can be misleading. One might argue, for instance, that Figure 6 underestimates WAFL's performance because the FAServer cluster has only 8 file systems, while the other servers all have dozens. On a per file system basis, WAFL outperforms the other file systems by almost eight to one. Furthermore, the FAServer uses RAID (which typically degrades file system performance substantially for the small request sizes characteristic of NFS), whereas the other servers do not use RAID.

On the other hand, one might argue that the benchmark overestimates WAFL's performance, because the entire FAServer is designed specifically for NFS, and much of its performance comes from NFS-specific tuning of the whole system not just to WAFL.

Given WAFL's special purpose nature, there is probably no fair way to compare its performance to general purpose file systems, but it clearly satisfies the design goal of performing well with NFS and RAID.

6. Conclusion

WAFL was developed, and has become stable, surprisingly quickly for a new file system. It has been in use as a production file system for over a year, and we know of no case where it has lost user data.

We attribute this stability in part to WAFL's use of consistency points. Processing file system requests is simple because WAFL updates only in-memory data structures and the NVRAM log. Consistency points eliminate ordering constraints for disk writes, which are a significant source of bugs in most file systems. The code that writes consistency points is concentrated in a single file, it interacts little with the rest of WAFL, and it executes relatively infrequently.

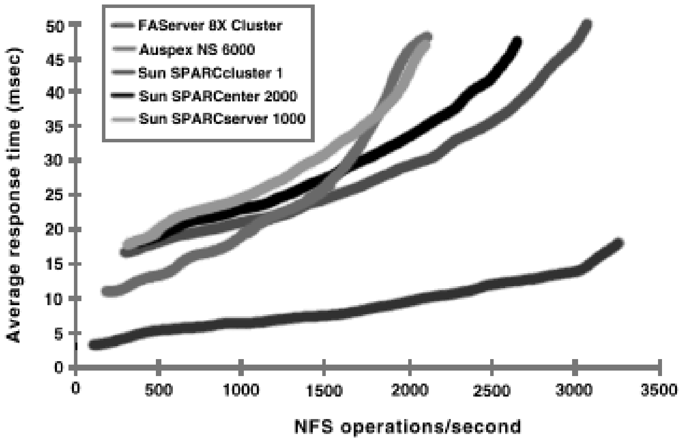

Figure 6: Graph of SPECnfs_A93 operations per second. (For clusters, the graph shows SPECnfs_A93 cluster operations per second.)

More importantly, we believe that it is much easier to develop high quality, high performance system software for an appliance than for a general purpose operating system. Compared to a general purpose file system, WAFL handles a very regular and simple set of requests. A general purpose file system receives requests from thousands of different applications with a wide variety of different access patterns, and new applications are added frequently. By contrast, WAFL receives requests only from the NFS client code of other systems. There are few NFS client implementations, and new implementations are rare. Of course, applications are the ultimate source of NFS requests, but the NFS client code converts file system requests into a regular pattern of network requests, and it filters out error cases before they reach the server. The small number of operations that WAFL supports makes it possible to define and test the entire range of inputs that it is expected to handle.

These advantages apply to any appliance, not just to file server appliances. A network appliance only makes sense for protocols that are well defined and widely used, but for such protocols, an appliance can provide important advantages over a general purpose computer.

Bibliography

[Astrahan76] M. Astrahan, M. Blasgen, K. Chamberlain, K. Eswaran, J. Gravy, P. Griffiths, W. King, I. Traiger, B. Wade and V. Watson. System R: Relational Approach to Database

Management. ACM Transactions on Database Systems 1, 2 (1976), pp. 97-137.

[Chutani92] Sailesh Chutani, et. al. The Episode File System. Proceedings of the Winter 1992 USENIX Conference, pp. 43-60, San Francisco, CA, January 1992.

[Hitz93] Dave Hitz. An NFS File Server Appliance. Network Appliance, Inc.

[Lyon89] Bob Lyon and Russel Sandberg. Breaking Through the NFS Performance Barrier. SunTech Journal 2(4): 21-27, Autumn 1989.

[McKusick84] Marshall K. McKusick. A Fast File System for UNIX. ACM Transactions on Computer Systems 2(3): 181-97, August 1984.

[Ousterhout89] John Ousterhout and Fred Douglis Beating the I/O Bottleneck: A Case for Log-Structured File Systems. ACM SIGOPS, 23, January 1989.

[Patterson88] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). ACM SIGMOD 88, Chicago, June 1988, pp. 109-116.

[Sandberg85] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Implementation of the Sun Network File System. Proceedings of the Summer 1985 USENIX Conference, pp. 119- 30, Portland, OR, June 1985.
