Technical Report

High-File-Count Environment Best Practices

Shree Reddy, NetApp
January 2012 | TR-3537

MANAGING BILLIONS OF FILES USING DATA ONTAP AND WAFL

The goal of this report is to facilitate the successful deployment of NetApp® storage appliances in environments with many millions or even billions of files. We provide an overview of best practices and recommendations for deploying and managing Data ONTAP® storage systems for high-file-count environments. To place the best practices in context, we first present the theory and background necessary for an in-depth understanding of how Data ONTAP and the WAFL® file system operate in these environments. We conclude this technical report with a practical sizing methodology to approximate a working set and select an appropriate NetApp platform for deploying the target workload.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Many applications store data objects as individual files and directories. At one time, this approach was chosen for convenience and to make application development simpler. As the roles of these applications and the demands on them grow, the set of files and directories created and managed expands into the tens or even hundreds of millions.

High-file-count environments have their own particular set of behaviors that affect many aspects of storage. This technical report explores the contributing causes of those behaviors and provides practical suggestions for successfully deploying NetApp storage running Data ONTAP and WAFL (Write Anywhere File Layout) with applications in high-file-count environments.

1.1 DEFINITION OF HIGH-FILE-COUNT ENVIRONMENTS

Before we dive into the details, let's define what we mean by high-file-count environments. Any application or NetApp system in which one or more of the following conditions is true is considered a high-file-count environment:

- Single directories frequently contain more than 50,000 files each.
- Single volumes contain more than 10 million files each.
- Millions of active files have long file names and/or deep directory trees.

This technical report can help to manage these environments.

1.2 CHALLENGES

Managing high-file-count environments holds a number of challenges, mostly because of the complexity of how practical limitations in the underlying hardware interact with Data ONTAP and WAFL implementation choices.

This technical report helps you understand the following challenges.

HOW FILE LAYOUT AFFECTS PERFORMANCE

File layout affects performance in these ways:

- Inode and directory block formats: How WAFL organizes file and directory metadata and translates file names to blocks on disks that hold the actual data
- Random disk access patterns: The operations required to access the metadata and data from disk, and related implications for performance and designing practical solutions

CAPACITY PLANNING FOR HIGH-FILE-COUNT ENVIRONMENTS

Planning for high-file-count environments is different from planning for environments in which the main consideration is total throughput and capacity. It's also necessary to consider:

- Planning for additional metadata: Make sure that the in-memory and on-disk space for metadata is taken into account when figuring out which storage platform to purchase or how much space will be available for file data.
- Approximating working set size: The working set should fit in memory for the best performance. In the context of a storage system, the working set can be considered as a collection of data and metadata blocks that are either currently being accessed by all active applications and users or that will be accessed within a short period of time. The working set depends on application behavior, and the size of the working set depends on the data and metadata involved.

TUNING DATA ONTAP OPTIONS

Data ONTAP provides several tunable parameters to help optimize performance and resource utilization in high-file-count environments. The challenge is knowing when to adjust these parameters and how to set their values based on the expected workload.

OPERATIONAL CONSIDERATIONS

Other aspects of administering storage in a high-file-count environment also come into play. Understanding how these can be addressed is another challenge that this technical report addresses.

1.3 OVERVIEW OF THIS TECHNICAL REPORT

This technical report is a high-level introduction to how WAFL manages files and directories, including examples of scenarios relevant to high-file-count environments. The first part of the report presents the theory and background that are necessary for in-depth understanding but that are not essential for using the recommendations. The second part provides an overview of best practices and recommendations for deploying and managing Data ONTAP storage systems for high-file-count environments. The final section is a practical sizing methodology to approximate a working set and to select the appropriate NetApp platform for the workload.

This report is focused on flexible volumes in Data ONTAP 7.0 and later. Prior versions of Data ONTAP and traditional volumes behave similarly, but the details and limits are slightly different. Where appropriate, both the FlexVol® volume limits and traditional limits are mentioned.

2 BACKGROUND AND THEORY OF OPERATION

2.1 WAFL

Data ONTAP stores data on volumes formatted using the WAFL file system. Initially deployed in 1993, WAFL continues to evolve and includes a long list of potential future improvements.

WAFL stores metadata in files, which allows it to write metadata blocks anywhere on disk. This is the origin of the name WAFL: *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. WAFL additionally makes it easy to find areas of the volume where the full stripes can fit, since the blocks can go anywhere. It also offers more flexibility than traditional file-system implementations, which require much of the metadata to reside at fixed locations on disk as well as frequent seek operations to update data in place for some workloads.

For an in-depth discussion, see TR-3002: File System Design for an NFS File Server Appliance and TR-3001: A Storage Networking Appliance.

INODES

An *inode* is a collection of information about a storage object, usually a file or directory. Inode information is held in the *inode file,* which is a hidden system file contained in every volume. Each chunk of inode information is currently 192 bytes long and is allocated on disk in 4KB blocks. Inodes are referenced by a number, which is the offset of the inode information in the inode file. In other words, the information for inode 96 starts 96*192 bytes into the inode file and covers the next 192 bytes.

The inode holds information that includes time/date stamps, size, UNIX® permissions, and so on, as well as either the file data (if the file contains less than 65 bytes) or a reference to where to find a block holding either data or references to other blocks, such as indirect blocks.

Each WAFL inode contains 8 block pointers for 32-bit flexible volumes, 5 block pointers for 64-bit flexible volumes, and 16 block pointers for traditional volumes to indicate which data blocks belong to the file. WAFL data blocks are currently fixed at 4KB. All of the block pointers in a WAFL inode refer to blocks at the same level. Thus inodes for files smaller than 32KB for 32-bit flexible volumes (20KB for 64-bit flexible

volumes, 64KB for traditional volumes) use the block pointers in the WAFL inode to point to data blocks. Inodes for files smaller than roughly 16MB for 32-bit flexible volumes (5MB for 64-bit flexible volumes, 64MB for traditional volumes) point to indirect blocks, which point to actual file data. Inodes for larger files point to double-indirect blocks, or triple-indirect blocks for very huge files. For very small files (64 bytes or less), it is more efficient to store the file data in the inode itself in place of the block pointers.

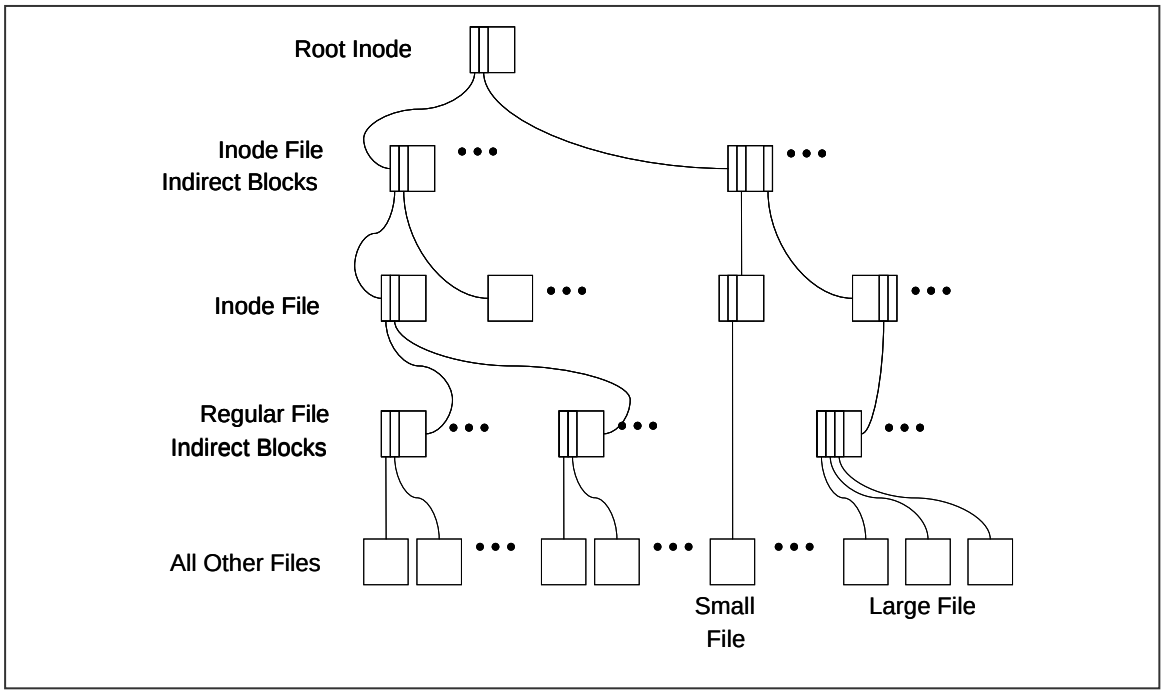
Blocks for the inode file are allocated from available space in the volume and decrease the number of blocks available for other data. Data ONTAP allows the administrator to grow the inode file to make room for additional files in the volume. Data ONTAP currently allows for no more than 1 inode per 4KB of disk space in the volume up to a maximum of 2 billion inodes. This is a safety and usability feature to prevent the inode file from growing too big. To see current inode usage for a volume, the appliance administrator can use the `df –i` command (`vol show –field files/vol show –field files-used` for Cluster-Mode).

In large-file-count environments, files tend to be small, and it's not unusual to find average file sizes below 4KB. Remember that if the file is 64 bytes or smaller it goes directly into the inode; otherwise, Data ONTAP allocates a WAFL block on disk, which is a minimum of 4KB in size. So unless the average file size is below 64 bytes, the default inode limit makes sense.

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 the figure below. 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. The leaves of the tree are the data blocks of all the files and directories. WAFL directories are treated at some level just like regular files, but they are managed and interpreted by WAFL instead of user applications. Metadata files and regular client-visible files are stored in separate inode files.

Figure 1 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 the WAFL write-anywhere rule is that the block referencing the root inode must live at a fixed location on disk where WAFL can find it.

Figure 1) A more detailed view of the WAFL tree of blocks



DIRECTORY BLOCKS

Directories have a special type (specified in the inode) so that Data ONTAP treats them appropriately. WAFL directories currently always take at least one 4KB WAFL block on disk:

$$4KB = \left(2bytes \times 128entries = 1536bytes\right) + \left(6bytes \times 160chunks = 2560bytes\right)$$

Each directory block is broken into two primary portions: an array of *entries* and an array of *name chunks.* The entry array for each directory block contains 128 rows, each currently 12 bytes long. The rest of the WAFL block contains the array of 160 name chunks, each 16 bytes long. Entries contain a file ID, a generation number, and a pointer to related name chunks. If the NFS name of a storage object in the directory is longer than 16 characters, it uses multiple name chunks from that directory block.

Directory entries often have three names: one for NFS, another for DOS 8.3 compatibility, and a third for Unicode naming. Unicode and NFS can't be merged because NFS is case sensitive and CIFS is not. If a file has multiple names, each name uses a separate name chunk. Some Unicode character sets use two bytes per character, in which case the Unicode name will need a name chunk per eight characters. Once all the name chunks in a directory block are used, WAFL uses another directory block even though there are still entry slots available. The same holds true for using up all the name slots for NFS environments with short names. Finally, two files with the same name are not allowed in the same directory.

In cases in which the directory information is large, indirect blocks are used to refer to directory blocks or other indirect blocks. Just as in a regular file, directory information is kept only in the leaf blocks. Directory information kept on disk currently cannot shrink. If you fill your directory with 300,000 files and then remove them all, the directory size (number of blocks used by the WAFL file representing that directory) is not reduced. However, if you then continue to add files to that directory, the existing directory structure is filled rather than additional ones being created.

2.2 DATA PROTECTION

RAID

One frequently overlooked issue in high-file-count environments is what happens to performance when a disk in a RAID group fails. Using RAID 4, when a single drive fails (single-parity drive), the reconstruct is done at high priority to minimize the window of time without parity protection. This keeps the disk drives and Data ONTAP very busy and consequently reduces resources available to services that a WAFL file requests. Since the disk access pattern for high-file-count environments tends to be more random in nature, response times increase very rapidly and may result in unacceptable performance.

Data ONTAP has a number of mechanisms to reduce the likelihood of this situation. The most effective option is to use RAID-DP® technology for high-file-count volumes. Because RAID-DP has two parity disks, the RAID group still has a level of parity protection after a single disk failure. This allows Data ONTAP to perform disk reconstructs in the background at a lower priority and greatly lessens the impact of disk failures on performance.

To learn more about Data ONTAP features for data protection and resiliency, see TR-3437: Storage Best Practices and Resiliency Guide.

FILE SYSTEM CONSISTENCY

WAFL uses a consistent on-disk format that should never require additional levels of consistency checking or repair. In particular, and unlike most UNIX and Windows® file systems, running a check is not required after power failures or unplanned reboots. With many tens of thousands of deployed systems, NetApp has occasionally encountered situations (for example, NVRAM battery failed during extended power outage, RAID 4 failed disk reconstruct plus ECC media error) in which a certain level of file-system checking and repair helped. Our goal is for customers to never need to run WAFL consistency checks, but the reality is that it is occasionally necessary.

There are currently two flavors of consistency checking for WAFL: `WAFL_check`[1] and `WAFL_iron`. `WAFL_check` is an offline command that is run from the interactive boot menu before the appliance becomes accessible to clients. `WAFL_iron` runs after the appliance finishes booting and runs in parallel with client access requests. The benefit of `WAFL_check` is that the system is in a well-known state and not dealing with additional load. This allows the command to complete much more quickly. `WAFL_iron` can run while the appliance is serving data, but the background load may noticeably degrade performance. In Data ONTAP 8.1, `WAFL_check` is no longer supported and is replaced by running `WAFL_iron` with the "`-o`" option.

When consistency checking is required, Data ONTAP goes through the metadata and determines that all the addressed files and directories are consistent. It also attempts to locate any files that look valid but that are not linked from anywhere, and places them where the administrator can manually move them to the appropriate location. These operations are performed at the file and directory level, and in the context of a high-file-count environment may take a very long time to complete. They are likely to exceed even very long maintenance windows. `WAFL_iron` helps in these situations, but the appliance may run with degraded performance for days while `WAFL_iron` completely checks the volume in the background.

When planning high-file-count environments, administrators should take resiliency into consideration. This means using RAID-DP for all high-file-count volumes. It also means being careful about providing consistent power, performing preventive maintenance so that NVRAM batteries are able to hold a full charge, and having a disaster management plan to deal with extended power outages.

NDMP AND QTREE SNAPMIRROR

NDMP (7-Mode and Cluster-Mode) and Qtree SnapMirror® (QSM, 7-Mode) technology are both logical replication mechanisms. They dump and restore information in terms of files and directories. That makes them generally more flexible and appropriate in situations in which the more efficient block-based mechanisms don't make sense to use; for example, they replicate only a small part of a large volume, back up to tape via NDMP, and so on.

Data ONTAP builds a plan for which files and directories will be transferred before the actual data movement begins. The order in which files and directories are written to the destination is important. For example, the NDMP dump must be sequentially written to tape in a way that allows the restore process to obtain all the catalog information it needs before actually restoring the data.

The transfers are planned in multiple phases:

- The first phase builds a map of files that need to be dumped.
- The second phase builds a map of the directories to dump.
- The third phase dumps the directories to the destinations.
- The fourth phase dumps the files to the destination.
- The fifth phase dumps the ACLs to the destination.

The destination can simultaneously restore the data it receives from the source (for example, QSM, NDMP copy, and so on) or just sequentially write it to tape.

The process of building the file and directory maps can take a long time for high-file-count environments. It is common for the Data ONTAP appliance to take many hours or even several days to build the transfer maps for environments with hundreds of millions of files. If you've ever attempted an NDMP backup or QSM transfer on millions of files, you've probably noticed that it can take many hours before the actual data transfer begins.

---

[1] This is sometimes referred to as "WACK," a play on the UNIX file system check `fsck`.

This processing happens in the background, and there are *mechanisms to minimize any impact* it may have on production workloads. However, the data may be changing rapidly and that means that even incremental backups take a long time to build the catalog. Also, there may be insufficient time for the setup and transfer to complete. It's not uncommon for high-file-count environments to require different backup mechanisms. To minimize the impact and help maintain even resource utilization, QSM destinations should be spread across as many destination controllers as possible.

Volume SnapMirror (VSM) is the most useful choice for replicating high-file-count volumes. It has all the benefits and advantages of SnapMirror over NDMP. Because the replication is block based, there is no per-file processing involved. Data ONTAP does not need to build a catalog of files, and the block transfers can begin immediately. VSM may be the only viable option if the replication needs to be complete before a certain replication window closes or another replication is scheduled to begin.

Both SnapMirror and NDMP are large subjects that we can't cover in depth in this report. For more information, see TR-3445: SnapMirror Best Practices Guide.

3 HIGH-FILE-COUNT CHALLENGES IN DEPTH

One of the best ways to learn about a complex subject in depth is to look at example scenarios. This section provides some useful examples that illustrate key points about high-file-count environments.

3.1 LOTS OF FILES IN A SINGLE VOLUME (INODE FILE SIZE)

An inode is required for each storage object (file, directory, link, and so on) referenced by the volume. These inodes are stored on disk in a system file that uses 192 bytes per inode. To calculate the amount of disk space required to store the inode information, multiply the estimated number of files and directories by 192 bytes. For example, a volume holding 5 million files requires approximately 915MB of inode information. In addition, if those 5 million files are all part of the working set, then the controller also uses that same amount of system memory to cache the inode information.

Memory usage per file is somewhat more complicated to estimate. In addition to the inode information, Data ONTAP stores other metadata associated with each file. This metadata is used to speed up operations and enable consistency. The exact amount of memory used per file depends on the controller platform. As an estimate, assume that Data ONTAP requires at least *700 bytes* of controller memory for each file in the working set. These 700 bytes include the 192 bytes for the inode, plus the additional metadata.

3.2 LOTS OF FILES IN A SINGLE DIRECTORY

Directories are used to logically organize files, but they sometimes grow very large due to quirks of the application. This section describes how Data ONTAP stores directory information and scales it as the number of files per directory increases.

NO INDIRECT BLOCKS

Let us consider a single inode representing a directory in a 32-bit FlexVol volume. This directory inode can point to 8 blocks (32K) worth of directory information. Suppose that this is an NFS environment and the file names are short, so only a single name chunk is used per file. This means that for each block all 128 entries are used, and there are 32 unused name chunks per block. The 8 blocks allow us to store 1,024 file names in the directory using a single inode with no indirect blocks. Because two of the names are "**.**" and"**..**" the directory can contain 1,022 files.

Now suppose that the directory is referenced by both CIFS and NFS, and the file names are still less than 16 characters. We now need to use 3 name chunks per file. That gives us the ability to reference 53 files per directory block, with 75 name entries and a single name chunk left empty. With 8 blocks addressable by the inode, the directory can hold 424 files, including "**.**" and "**..**". If there are more files than this, we will need to use a layer of indirect blocks to reference the files.

Since the directory blocks were all referenced with the original inode for the directory and used no indirect blocks, the total amount of space used by this directory is 8 blocks (32KB).

SINGLE LAYER OF INDIRECT BLOCKS

Now let's see how much we can store using a single layer of indirect blocks. The inode for the directory points to indirect blocks, and these blocks in turn point to blocks holding directory information (name entries and chunks). The inode has a special type that tells Data ONTAP that it represents a directory, so that we know to interpret all those data blocks as a single directory.

With 32-bit flexible volumes, the inode can point to 8 indirect blocks, and each indirect block can hold pointers for 510 blocks of directory information. With 8 indirect blocks, that works out to 4,080 blocks of directory information. With 64-bit flexible volumes, the inode can point to 5 indirect blocks, and each indirect block can hold pointers for 255 blocks of directory information. Traditional volumes can hold pointers for 16 blocks in the inode and 1,024 blocks per indirect block. Let's use the same calculations and assumptions as in the previous case. In the NFS-only case for a 32-bit flexible volume, that means we can reference $(4080 \times 128) - 2 = 522{,}238$ files with a single layer of indirection. In the CIFS/mixed case, that means we can reference $(4080 \times 53) - 2 = 216{,}238$ files with a single layer of indirection. If there are more files than this, we will need to go to a second level of indirection.

This directory would use 4,080 directory blocks, plus 8 indirect blocks on disk. With 4KB blocks, that works out to 16320KB (directory blocks) plus 32KB (indirect blocks) of disk space.

DOUBLE LAYER OF INDIRECT BLOCKS

We can repeat the same calculations for double-indirect blocks. The actual arithmetic is left as an exercise for the reader. In the NFS-only case, this works out to a little over 266 million files, and in the CIFS/mixed case this works out to about 110 million files. The directory information (8 double indirect, 4,080 single indirect, plus 2,080,800 directory blocks) would require a little less than 8GB of space on disk and 10GB in memory. This is larger than available memory in several NetApp controllers today, meaning that directories that large cannot be deployed on that hardware. Even when it is physically possible to deploy them, directories of that size are not very practical, and the application should be rewritten to take better advantage of subdirectories.

TRIPLE LAYER OF INDIRECT BLOCKS

It is theoretically possible to use triple-indirect blocks to address billions of files in a single directory. However, depending on the underlying controller platform, it may be impossible to actually run that configuration, and definitely impractical. It's much better to simply design the application directory layout to have fewer files per directory.

INODE FILE AND MAXIMUM NUMBER OF FILES PER VOLUME

Remember that Data ONTAP also uses a 192-byte inode per file. The inodes require space on disk and memory, and also reduce the amount of space for file data available in the volume. For a 16TB volume, assuming that we have a little over 9TB of usable space to work with, we subtract the space used for directory blocks and divide by 192 bytes to calculate how many inodes the volume can hold. There is no space left for file blocks, but each inode can hold up to 64 bytes of file data, so this may work for very small files.

Based on these calculations, one conclusion is that placing many billions of files in a single directory or volume is theoretically possible. However, it's not necessarily practical in terms of physical resources and what can be kept in the actual files. Also, Data ONTAP manages the maximum number of inodes associated with a volume using the `maxfiles(7-Mode)/vol modify -files` (Cluster-Mode) tunable parameter. Data ONTAP currently limits raising the maximum inode count to at most 1 inode per 4KB of usable volume size (with a default of 1 inode per 32KB) up to a maximum of 2 billion inodes. So for 9TB, 1 inode per 4KB works out to about 2.4 billion files; hence, the inode count is capped at 2 billion. For more discussion on the maximum number of inodes, see "Adjustable limits in WAFL," later in this paper.

MAXIMUM NUMBER OF SUBDIRECTORIES PER DIRECTORY

Prior to Data ONTAP 8.1, each WAFL directory can have at most 99,998 subdirectories. The reasons for this have to do with how hard links work and are implemented. A hard link is a mapping between a name and an inode number. The hard link appears as a file in a directory. The hard link mapping doesn't have to be unique, but names in any one directory must be unique. An inode number can be referenced by multiple names. Each reference adds to the link count for that inode. The inode link count is implemented as a 4-byte field, with a maximum value of 100,000. Hard links to a single inode can be in a single directory or in multiple directories, but they must be in the same volume.

Directories make extensive use of and are implemented using hard links. Hard links are used to show connectivity between parent and child directories in a directory tree. Each directory has a minimum of two entries, "`.`" and "`..`". The "`.`" reference is a hard link to the containing directory, and "`..`" is a hard link to the parent directory. A subdirectory is referenced via a hard link from the parent directory. Consider this: Each subdirectory has a hard link back to its parent directory, the inode for that parent directory can be linked at most 100,000 times, and every directory is created with 2 hard links. This means that each directory can have at most 99,998 subdirectories.

In Data ONTAP 8.1, there is no explicit limit on the number of subdirectories a WAFL directory can contain. The user can continue to create subdirectories as long as the `maxdirsize` and `maxfiles` values are not exceeded.

DIRECTORIES AS TEMPORARY WORKING SPACE

Directories on disk grow as files are added, but they don't shrink when files are deleted. Suppose that a directory is used as the temporary working space for computation or simulation that generates many thousands or millions of temporary files. These files are then collated and processed to generate a small set of final result files. The final files remain in the directory associated with the computation, but the temporary working files are deleted.

Assume that, at the peak, the working directory held 2 million files. This means that the directory structure took up approximately 64MB on disk. Now that the computation is concluded, the directory holds only a small number of files. If these files are placed in a new directory, that directory may be able to fit within 4K on disk. However, since the directory size never shrinks, the directory uses an unnecessary 65,532KB of space.

This implies a recommended practice to use space efficiently. In the situation just described, it's best to copy the final results to another directory and delete the original working directory to free up the unnecessary space.

3.3 LONG PATHS AND COMPLICATED NAMING

The path name to a target file is also a relevant performance factor in high-file-count environments. Data ONTAP needs to have a way of mapping from a file name to an inode on disk that holds the file data referenced by the name. Data ONTAP uses a caching mechanism to make repeated references to the same file efficient. System memory for the cache is set aside when Data ONTAP boots and the cache is sized to reference the maximum number of files that a particular platform can reference simultaneously. The size depends on physical memory on the platform and cannot be adjusted. Each name cache entry has a finite size, *currently set at 40 bytes*. This allows file names of up to 40 ASCII characters or up to 20 Unicode characters to be cached. Any file names longer than that do not fit in the cache and therefore must be resolved by following pointers.

LONG NAMES

WAFL limits file names to 255 characters. Long names can complicate performance in high-file-count environments in several ways. First, if the names are longer than can fit into a single name chunk in the directory block, then multiple name chunks must be used. This makes the directories use up more space on disk and in memory.

More importantly, the lookup name cache can hold only file names of a finite size (currently 40 bytes). File names longer than that need to be resolved again and again by WAFL. This can cause a significant slowdown in performance and increased CPU utilization. It's possible for applications that are otherwise perfectly tuned and configured to start performing badly just because the naming convention changed to one that exceeded the cache entry size.

VERY DEEP DIRECTORIES (8+)

File-name resolution happens for each component of the path. The more components (deeper directories), the more steps are required to find the inode for the leaf file. Very deep directories are a particular challenge for NFS because NFS resolves file/directory handles one subdirectory at a time. If you have a long path (for example, `foo:/vol1/aaa/bbb/ccc/ddd/eee/fff/ggg/hhh/h999834`) in which each leaf directory holds a million files and subdirectories, you first need to load all of the aaa directory into memory, resolve the handle for directory `bbb`, then load `bbb` into memory, resolve the handle for `ccc`, and so on. Each step of the NFS resolution happens over the network, with the inherently higher (than memory or disk) latencies that entails.

Also, each time a directory is loaded into memory for the lookup, it displaces a bunch of other information that the application may have been using. Plus there is the work of reading the directory data on disk and formatting it properly in memory for fast operations. For directories with a million files, that would be about 32 to 64MB each. If that process is repeated many times for each pathname, Data ONTAP will be kept busy just traversing directory trees.

VOLUME ACCESS MODE AND LANGUAGE

Volumes using NFS-only access and a non-Unicode language setting can use a single name chunk for file names shorter than 16 characters. All other combinations require the Unicode naming mechanism. Access mode and language for a volume can be adjusted dynamically. Once the change is made, the first CIFS or Unicode client access to a directory triggers a conversion process that changes all the names in that directory to use the new access and language scheme. Conversions can also be triggered when a directory is created and populated via NFS on a mixed-mode (NFS and CIFS) volume using non-Unicode language, and then accessed by a CIFS client.

The directory conversion must complete atomically, so no other user activity for the entire system is allowed until completion. For high-file-count environments with millions of files to convert, the operation can take a long time. This can be observed as significant periods of unresponsiveness and impact on service-level agreements (SLAs).

This behavior is a characteristic of current versions of Data ONTAP, and not much can be adjusted to limit the impact of directory format conversion. The best way to manage this facet of Data ONTAP is to plan accordingly and create the volume with the characteristics it might use eventually. If you know that the application will require a high-file-count environment and may eventually be accessed via CIFS and/or require Unicode file names, there are two volume options that can help: `create_ucode` and `convert_ucode`. When these options are turned on and directories are created using Unicode directory structures, any existing directories that are not already in that format are also converted on access. This reduces any surprises and can be touched during an access window to limit impacts on SLAs.

To turn on these options, use the standard syntax:
7-Mode:

```
vol options foo_vol create_ucode on
vol options foo_vol convert_ucode on
```

Cluster-Mode:

`vol modify -volume foo_vol -convert-ucode true` (only true for volumes that have been transitioned from 7-mode)

Cluster-Mode only supports the Unicode directory format. The only exception is if a volume has been transitioned from 7-Mode.

3.4 ADJUSTABLE LIMITS IN WAFL

Data ONTAP provides a number of adjustable limits that are set to provide a great out-of-the-box experience for most NetApp customers. Certain applications and environments benefit from adjusting these defaults, and the following two parameters most directly impact high-file-count environments: `maxdirsize` and `maxfiles`(7-Mode)`/vol modify -files`(Cluster-Mode).

MAXDIRSIZE

The `maxdirsize` parameter represents the largest possible directory (metadata and indirect blocks) in kilobytes. This parameter defaults to 1% of system memory, and for newer systems rarely needs to be adjusted. For example, a FAS3050 with 3GB of addressable system memory will support directories that hold over 400,000 CIFS files without changing the default `maxdirsize`. The setting is a per-FlexVol-volume setting (the global option has been deprecated), and applies to every individual directory in the FlexVol volume. All the file and subdirectory entries in a given directory are accounted for in `maxdirsize`. Files contained within subdirectories do not count against the `maxdirsize` of the directory.

If the parameter needs to be increased, limit the size to at most three times the default (3% of system memory). NetApp WAFL engineers strongly recommend this. Don't go above that number unless you are sure that only a few directories will be that large over the life of the application.

Also, make sure that all related volumes in the ecosystem (for example, the SnapMirror destination) have had the same tuned parameters and that the 3x guideline hasn't been exceeded for the second platform. For example, suppose that a FAS6070 controller is hosting a CIFS high-file-count environment with a default `maxdirsize` of about 325MB and the FlexVol volume in question has a SnapMirror destination on a FAS3050 controller with a default `maxdirsize` of 30.72MB. Imagine that the FlexVol volume holds a directory with about 1 million files (about 74MB of metadata). When Data ONTAP tries to create that directory on the SnapMirror destination, it notices that `maxdirsize` is exceeded and aborts the operation. By tuning `maxdirsize` to be 74MB, the SnapMirror operation can proceed. However, if the directory held 2 million files (about 150MB of metadata), increasing `maxdirsize` on the destination would violate the 3x guideline and NetApp strongly discourages this.

Remember that once a directory grows larger, it can never shrink. If you are tuning `maxdirsize` to allow very large directories, be vigilant to recover space from directories that have grown and then shrunk significantly.

In 7-Mode, the `maxdirsize` value can be tuned using `vol options maxdirsize`. In Cluster-Mode, you can set the maximum directory size in advanced mode `vol modify -volume <volname> -maxdir-size <size>`.

Can the `maxdirsize` be changed on the fly to increase the directory size when the volume is online?
Yes. You can change the `maxdirsize` on the fly to increase the directory size.

Can the `maxdirsize` value be decreased?

Yes. New entries will be disallowed in directories that are sized bigger than the `maxdirsize` value.

MAXIMUM INODE COUNT

As previously mentioned in "Directory blocks," the `maxfiles`(7-Mode)`/vol modify -files`(Cluster-Mode) parameter represents the maximum number of files and directories that can exist in a volume. There is a direct correlation between the `maxfiles` parameter and the number of inodes set aside for the volume.

By default, `maxfiles` is set to 1 inode per 32KB of volume size. When a volume grows, the inode count is increased automatically to MAXIMUM of (1 inode per 32KB of volume size, 33 million). This can be further increased to 1 inode per 4KB of volume size up to an enforced maximum of 2 billion inodes using the `maxfiles`(7-Mode)/`vol modify -files`(Cluster-Mode) command.

So irrespective of the volume sizes, WAFL restricts the inode count to a maximum of 2 billion inodes. The maximum inode count should be large enough to accommodate the total sum of files and directories in the volume, plus enough room for anticipated growth. Be sure that this increase is required, because any space set aside for inodes is taken away from space available in the volume.

The maximum inode count is associated with an increase in the inode file size to hold more inodes. If the maximum inode count is set to a high value but the inodes are not used, the inode file size does not increase and no extra blocks are allocated. So, setting the inode count to a high value does not automatically increase space consumption. Space is consumed only when inodes are used. If all the inodes up to the maximum inode count are used, then the space used is in fact not recoverable.

In releases prior to Data ONTAP 8.0, once the maximum inode count is increased the space associated with the increase can never be released (that is, the maximum inode count cannot be decreased).

In Data ONTAP 8.0 onwards, inode allocation happens from a smaller restricted range even though the maximum inode count is much higher. Once all the inodes in this range are used, WAFL dynamically increases the size of the current inode allocation range by a certain percent up to the maximum inode count. Consequently, with Data ONTAP 8.0, you can decrease the maximum inode count as long as the new value is greater than the current inode allocation range. The current size of the inode file is called the "capacity" and can be displayed in advanced mode via `maxfiles -capacity` (7-Mode/nodeshell cli in Cluster-Mode).

Will a high inode count affect performance?

Increasing the number of inodes affects system processes that traverse inodes in a volume. If data cannot be deleted and volume size cannot be increased, NetApp recommends increasing the inodes in a volume by 10-20% instead of setting an arbitrary high value. Increasing the size of the volume will also increase the number of available inodes.

Are inodes in snapshots counted against the maximum inode count?

No. Only inodes in the active file system are counted against the maximum inode count.

Best Practices
• The best way to provide great performance for applications that deal with many millions of objects is to design a solution that avoids dumping lots of files into a single directory or volume; for example, by using relational databases, keeping the directory tree flat and bushy (avoiding indirect directory blocks and large directory sizes), spreading the files across more resources (using several appliances instead of just one), and so on. If there is no way around a high-file-count environment, this section contains some practical recommendations.

3.5 GENERAL RECOMMENDATIONS

SELECT THE NETAPP APPLIANCE WITH THE MAXIMUM AMOUNT OF PHYSICAL MEMORY YOUR BUDGET CAN JUSTIFY

More memory helps with many areas of caching and performance. In particular, it allows larger directories and working sets to fit in memory. It takes many orders of magnitude longer to access data on disk than in memory. In more human terms, suppose it takes 20 nanoseconds (10^{-9} seconds) to access memory, and 20 milliseconds (10^{-3} seconds) to access disk. Disk access takes a *million* times longer than memory access. Imagine this: If accessing memory is equal to sitting through a 30-second TV commercial, then

accessing disk is equal to sitting through 347 *days* of commercials. Using the right amount of physical memory means that many more requests are completed orders of magnitude faster.

Since the entire directory needs to be loaded into memory for many operations, you also need to have sufficient memory to accommodate the entire working set of files and directories. For example, suppose there is a custom application that was naively written to use directories with approximately 10 million files each, and that about 100 of these directories are being used simultaneously by all instances of this application running across many different server hosts. That's a working set of about 1 *billion* files total.

We know from earlier examples that a directory in a 32-bit flexible volume using a single layer of indirection can address a little over 2 million files and takes up about 64MB of memory and disk for just the directory information. So a directory with 10 million files needs to use double-indirect directory blocks and uses approximately 5 x 64MB = 320MB of memory. A working set with 100 such directories would require approximately 32GB of memory just for directory information, and more memory to hold the data blocks associated with the files.

The NetApp FAS6070A has two controllers with 32GB of controller RAM each. If the directories in the previous example were evenly distributed across both the controllers, then all the directory information could fit in memory simultaneously. Assume that Data ONTAP uses approximately 1GB for itself on each controller. That leaves about 15GB per controller for data blocks needed by the application. Also, remember that you need to access the inodes for each file. The previous scenario would have approximately 179GB of inode metadata, which would need to be continually loaded into memory from disk as different files are accessed.

The previous example illustrates that it's possible to support such applications using Data ONTAP and NetApp's current line of controller hardware. However, it would probably be far more efficient to have those same appliances provide block-based LUNs via FCP or iSCSI, and to rewrite the application to use something like an Oracle® Database to store the 1 billion objects currently stored in files. The LUNs would use almost no controller memory to store directory information, leaving the entire memory to help accelerate access to data blocks.

USE MULTIPLE STORAGE APPLIANCES

As shown in the previous example, if the physical memory in a single controller is not sufficient to hold the working set, then you can divide the working set across multiple controllers. If more than two controllers are needed, or if all the data needs to be accessible through a single access point (mount or share), then consider Data ONTAP operating in Cluster-Mode, which provides up to 24 controllers in a single global namespace. With up to 96GB of physical memory per controller on higher-end controllers like the FAS6280, the system should be able to accommodate working sets with many billions of files.

USE MULTIPLE FLEXIBLE VOLUMES TO DISTRIBUTE DATA

Data ONTAP supports hundreds of volumes per controller. Use all of these volumes by distributing the high-file-count environment across as many volumes as practical. Certain tunings and options are available on a per-volume basis. By breaking up the total set of files across different volumes, it is possible to assign different levels of resources to different sets of files for optimal performance.

For example, using the approach just described it's possible to turn a single volume with 300 million files into about 300 volumes, each with about 1 million files. You can then assign different values of `maxdirsize`, `maxfiles`, `minra`, and `no_atime_update` to the different volumes as appropriate. You can also assign different FlexShare® tool (7-Mode only) priorities to more closely reflect the operational priorities of the application.

Different NetApp controllers can support different maximum volume counts. Please refer to the Storage Management Guide to view the maximum volume limits currently configured on the system.

Data ONTAP operating in Cluster-Mode can also add significant value by allowing the administrator to create a single namespace. The namespace ties all the volumes across all the controllers in the cluster into one hierarchical tree. To client systems and applications, the namespace looks like one giant volume with petabytes of capacity. Volumes look just like directories in the namespace, and they don't need to be referenced via a specific controller or export in the `/vol` directory (for example, `toaster:/vol/vol1`).

Data ONTAP operating in Cluster-Mode helps to distribute data elegantly and simplifies data management.

AVOID INDIRECTION TOWARD THE BOTTOM OF THE DIRECTORY TREE

Keep the directory depth relatively shallow (less than 5 is ideal) in order to minimize NFS multilevel WAFL seeks to resolve file and directory handles. If the subdirectory depths are greater than about 8 or 9, the environment will continually bump into corner cases in which multilevel file handle resolution and path name caching issues create poor performance.

At the bottom of the directory tree, try to keep the number of files per directory less than:

32-bit flexible volume: 1,000 (NFS only) and 420 (mixed)

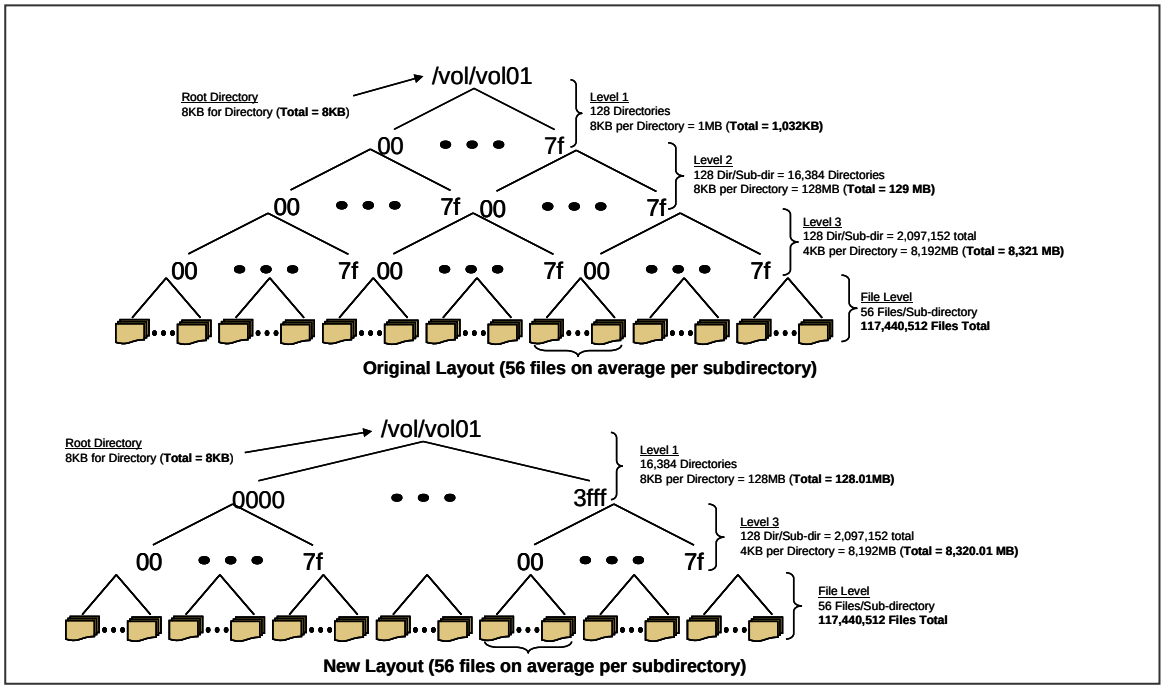
64-bit flexible volume: 640 (NFS only) and 265 (mixed)

Traditional volume: 2,048 (NFS only) and 848 (mixed)

As illustrated earlier in "Directory blocks," these are the numbers of files that can be addressed via direct reference to directory blocks from the directory inode. More files in occasional directories are not an issue, but the numbers cited are optimal based on current WAFL implementation.

With NFS, you can mount an intermediate directory and minimize the performance impact of the multilevel file handle resolution. For example, if you have a pathname like `/aaa/bbb/ccc/ddd/eee/fff/ggg/hhh/iii/jjj/kkk/lll/mmm` you can choose to mount an intermediate directory such as `ggg`.

Figure 2) Two directory layout schemes that address the same number of files and have approximately the same amount of metadata, but in which the new layout is more efficient than the original layout.



The amount of directory metadata is roughly the same regardless of whether the directory structure is wide toward the top or wide toward the bottom. For example, imagine an application that needs to have 2,097,152 ($2\times1024\times1024$) subdirectories. It's possible to lay these out in three levels so that there are 128 subdirectories per directory at each level. This layout uses a total of 8321MB of directory metadata. A better alternative is to create a two-level structure with 16,384 subdirectories at the first level and 128 subdirectories per directory at the second level. This layout uses 8320.5MB of directory metadata and

has the benefit of avoiding an additional level of directory structures. The two alternative schemes are illustrated in Figure 2.

REMEMBER THAT CLIENTS MAY NOT BE ABLE TO USE LARGE DIRECTORIES

Many host-based utilities do not scale with the number of objects in a directory. Try opening a Windows or Gnome file explorer in a subdirectory of 10,000 objects to get a better feel for the problem. Even if Data ONTAP can manage and serve volumes and directories with millions of files, users and applications on the client may not be able to use them effectively.

DELETE LARGE DIRECTORIES DURING NONCRITICAL TIMES

If you need to delete a large number of files in a single directory or a large directory, try to schedule the activity for off-peak hours or batch it into smaller jobs. For each file deleted, WAFL needs to decrease the reference count to the file, and if that file is not locked in any snapshot, delete the file by freeing the blocks, inode, and all the relevant metadata files. When a user requests deletion of all files within a single directory, WAFL deletes each file, updates the metadata, and updates the directory with this information. As the number of files increases, the performance of the appliance can be severely affected, since Data ONTAP frees blocks owned by each file within the directory. If all the files within a volume need to be deleted, destroying the entire volume is optimal.

In high-file-count environments, the `snap delete` command can also have a severe impact on performance. Imagine that all the files in a high-file-count directory have been deleted in the live volume but were locked by a snapshot. Most of the work associated with freeing the blocks was deferred when the files were originally deleted because the blocks were still in use by the snapshot. After the `snap delete` command returns, the free block scanner traverses the volume and marks the data blocks that were locked by the deleted snapshots as free. The associated load on the CPU and disk continue to be high until the scanner completes its run.

3.6 TUNE THE DISK SUBSYSTEM

In most cases, high-file-count environments generate a significant number of random read requests to the disk subsystem. Much of the activity is associated with fetching file and directory blocks that are not currently in memory. Random writes tend to be converted to sequential writes by WAFL and NVRAM, but random reads must be served by the disk. Over time, if the aggregate or traditional volume becomes close to full, it may be difficult to find contiguous free areas on disk to write updates. That results in disk seek operations to service writes as well as reads. All this adds up to the fact that the aggregate or traditional volume hosting a high-file-count environment must be able to service these random requests with low response times to maintain high performance. The following suggestions can help the disk subsystem to perform optimally.

USE HIGHER-RPM DISKS

Higher-RPM disks are able to service many more random I/O requests per second than lower-RPM disks (at the same latency level). Therefore an aggregate composed of 15K RPM disks performs much better than an aggregate that uses the same number of 10K RPM disks.

I/Os per second depend much more on the number of disks than on the size of the disk. Most high-file-count environments consist of relatively small files, and do not require much storage space. By using disks of smaller capacity but higher RPM, it's possible to configure a storage solution that provides better performance without significantly increasing cost.

CONFIGURE FEWER WIDE AGGREGATES

A related best practice is to use flexible volumes and configure a smaller number of wide aggregates, allowing more disks to be shared by multiple volumes. In most cases, not all volumes sharing an aggregate are active at the same time, which allows the active volumes to take advantage of the disk that would otherwise be sitting idle, hosting the less active volumes.

Using a single large aggregate also has the advantage that all the unused space is shared among all the volumes. That increases the chance of finding a large contiguous stretch of available space when WAFL needs to write to disk and therefore improves write performance. It also makes it possible to intelligently manage free space in the aggregate, and to make that space available to volumes that need it most.

USE RAID-DP EXCLUSIVELY

As mentioned previously, any disk failure in a RAID 4 group triggers reconstruction at high priority. Data ONTAP spends the majority of its time trying to rebuild the lost data onto a spare disk, and application access to all the volumes is degraded. Also, using RAID-DP significantly reduces the chances of needing to run WAFL consistency checks to recover from ECC media errors during disk reconstruction.

Let's take a closer look at reconstruction impact. Because high-file-count environments perform many random disk operations, they are sensitive to the response time of the disk subsystem. Latency increases exponentially with load, so it would be typical to see response times go from 1 millisecond to 30 milliseconds or more in cases in which there is a heavy background load.

Suppose that we needed to read in a directory with 2 million files (16,384 blocks of 4K each on disk). If in the worst case there were a disk seek for every 4 blocks read, it would take approximately 4 seconds to read the directory from disk at a 1 millisecond average response time. With a 30 milliseconds average response time, it would take over 2 minutes to read in the same directory.

By using RAID-DP, Data ONTAP has the luxury of running reconstruction at medium or even low priority. That still significantly increases the response time for the affected RAID group, but the impact on performance is much less. First, because fewer CPU cycles are used for reconstruction, all the volumes are less affected. Second, because the plex/aggregate with the failed disk is not reconstructing as aggressively, more disk bandwidth is available to serve application data.

3.7 CONFIGURE WAFL TO IMPROVE PERFORMANCE

RUN TRADITIONAL VOLUMES AND AGGREGATES BELOW 80% OF CAPACITY

The WAFL file system uses free space to serve disk writes. That means it needs to find space free whenever it has something to write. If the traditional volume or aggregate is mostly full in a high-file-count environment, chances are that the free space is not contiguous and is scattered throughout the volume. This induces disk seeks during write operations, further decreases performance, and increases the chance of back-to-back consistency points (which it is best to avoid for a number of reasons). This is not an issue for flexible volumes, since you can resize dynamically as needs and usage change and they draw on a pool of free blocks within the larger aggregate.

WAFL reserves some free space for itself when it publishes the amount of usable space available for file storage (WAFL reserve). However, the default amount (10%) may not be sufficient for high-file-count environments because the many small files being created, updated, and deleted tend to fragment free space over time. As a best practice, keep unused about 20% of the traditional volume or aggregate to provide more stretches of contiguous free space. This can be accomplished by simply monitoring free space in the volume.

CONSIDER SETTING NO_ATIME_UPDATE

Data ONTAP tracks a number of standard items of metadata for each file in a WAFL volume. One of these items is the file access time. Each time a file is accessed, the time is by default recorded by updating the associated inode. The access time can then be retrieved by various tools (for example, `ls -lut`) and may be useful for some applications. For most applications, it doesn't really matter when the file was last accessed. Yet by default Data ONTAP must still update the inode each time the file is accessed. For high-file-count environments, this can create a huge amount of wasted effort. In addition, it can create a significant amount of write activity for workloads that are expected to be read-only.

The administrator can tell Data ONTAP to skip updating the access time for all the files on a particular volume by using a volume option called `no_atime_update`; for example, `vol options foo_vol`

`no_atime_update on` (7-Mode)/ `vol modify –volume foo_vol atime-update` (Cluster-Mode). This can significantly increase performance for many applications in high-file-count environments.

CONSIDER SETTING MINRA

By default, Data ONTAP aggressively reads additional data from the volume. The assumption is that WAFL turns temporal locality into spatial locality, so any blocks neighboring the requested block are likely to be used in the near future. This is called "read-ahead," and for most workloads it improves performance.

If the blocks on disk are associated with millions of tiny files that are truly written and read randomly, then the underlying assumptions behind read-ahead are wrong and the resources associated with reading the additional data are wasted. This includes not only the extra work on the part of the disks and controller, but also the WAFL buffers used to store the blocks in memory. Memory can be a precious commodity for high-file-count environments, so it's best to use it as wisely as possible.

This option controls how many blocks are prefetched at each read operation.

The administrator can tell Data ONTAP to read only the bare minimum and avoid read-ahead. In 7-Mode, this is done by using a volume option called `minra;` for example, `vol options foo_vol minra on`. In Cluster-Mode, this option can be set by `vol modify –volume foo_vol –min-readahead true`. This can significantly increase performance for some applications in high-file-count environments. However, `minra` may actually decrease performance for other applications. The key is whether there is significant temporal or spatial locality for data on disk. If there isn't, then turning off read-ahead is likely to help. It's difficult to know this about an application with any certainty, so the best practice is to use the default setting (off) and only experiment with `minra` if performance impact is observed.

3.8 BACKING UP HIGH-FILE-COUNT ENVIRONMENTS

In backing up high-file-count environments, block-based data protection mechanisms are preferred over file-based mechanisms.

SNAPMIRROR

For disaster recovery, use volume SnapMirror (VSM). Avoid using Qtree SnapMirror (QSM, 7-Mode only) for high-file-count environments. As *mentioned previously,* volume SnapMirror works by transferring physical blocks starting from the first block in the volume and proceeding sequentially until it transfers the last block. VSM startup time is low and remains low in the presence of high-file-count environments. Even if more data needs to be replicated than with QSM, the saving in setting up and dumping time may be worth it.

NDMP

For tape backup, the only practical solution for high-file-count environments is to use SnapMirror to tape (SM2T), which works similarly to VSM. It also starts by transferring the physical blocks of the volume from first to last. SM2T startup time (the time you issue the command to the time that blocks start streaming out to tape) is low and remains low in the presence of high-file-count environments. Also, SnapMirror to tape is significantly faster (~10x) for driving tape than any other traditional backup tool because it is not subject to the start/stop reading and writing of small files.

SnapMirror to tape is not to be confused with third-party software backup package like Veritas™ NetBackup™, BakBone, or CommVault. These are full-featured backup suites. Some backup products, such as *BakBone,* provide additional plug-ins that integrate with NetApp appliances to manage SM2T functionality.

MAKE SURE THAT MAXDIRSIZE AND MAXIMUM INODE COUNT SETTINGS ARE COMPATIBLE ON SOURCE AND DESTINATION

As mentioned in "Adjustable Limits in WAFL," be careful to take `maxdirsize` and maximum inode counts into consideration when using either block-based or file-based replication mechanisms. As

mentioned earlier, it's possible to have a high-file-count volume as a replication source that cannot be created on the destination because of the `directory size or inode count` limit. This is true for both file-based mechanisms (QSM, NDMP) and block-based mechanisms (VSM).

These parameters get a default value based on platform capabilities. If replication happens between a larger and a smaller platform, the smaller platform may not be able to serve as the destination. The same issue may also arise if the limits have been raised on the source but are set lower at the destination.

4 FLASH CACHE

Flash Cache reduces data access latency and eliminates disk I/O through intelligent caching of recently read user data and metadata in the storage controller. Flash Cache acts as a second-level file system cache. It has the following modes of operation:

1. Metadata + User Data (default)
2. Metadata only
3. Metadata + User Data + Lopri

Flash cache actually uses some system memory to maintain information, thus reducing the amount of memory available for the necessary metadata operations (approximately 1GB per 256GB of cache).

For high-file-count environments, configure system controllers with Flash Cache wherever possible using the default setting of metadata and normal user data. Although Flash Cache can help improve the performance of high file-count environments, it should not be considered as a substitute for main memory requirements nor as additional system memory.

5 USING FLEXSHARE (7-MODE ONLY) IN HIGH-FILE-COUNT ENVIRONMENTS

FLEXSHARE OVERVIEW

This subsection provides a short overview of FlexShare functionality. If you are familiar with FlexShare, skip ahead to the next subsection, which focuses on best practices for applying FlexShare to high-file-count environments.

FlexShare is a Data ONTAP feature that provides control of service for a storage system. Using FlexShare, customers can confidently consolidate different applications and datasets onto a single storage system. FlexShare gives administrators the control to prioritize system resources based on how critical they are to the business or application.

FlexShare prioritizes processing resources for key services when the system is under heavy load. It should not be confused with quality of service (QoS), a term that implies resource *guarantees*. FlexShare does *not* provide guarantees on the availability of resources or on how long particular operations will take to complete. Instead, it provides a priority mechanism to give preferential treatment to higher-priority tasks. The one guarantee that FlexShare makes is that all legitimate requests will eventually be processed. This prevents requests with low priorities from getting starved and staying in the system indefinitely.

FlexShare provides storage systems with the following key features:

- Relative priority of different volumes
- Per-volume user versus system priority
- Per-volume cache policies

These features allow storage administrators to tune how the system should prioritize system resources in the event that the system is overloaded.

FlexShare is available on systems running Data ONTAP 7.2 and later. To learn more about FlexShare, see TR-3459: FlexShare Design and Implementation Guide.

5.1 TECHNICAL DETAILS

This subsection examines the most relevant FlexShare technical concepts.

WAFL MESSAGES

A WAFL message is the most rudimentary form of a WAFL operation. For example, Data ONTAP translates an NFS read or write operation to an individual WAFL read or WAFL write message. Similarly, system operations are translated into individual WAFL internal messages.

Each WAFL message is marked with an origin (*user* or *system*), which is used to determine whether the WAFL message was originated as a direct user request or an internal system task.

PROCESSING BUCKETS

FlexShare maintains different processing buckets (priority queues) for each volume that has a configured priority setting. FlexShare populates the processing buckets for each volume with WAFL messages as they are submitted for execution. The processing buckets are used only when the FlexShare service is on; when the FlexShare service is off, all WAFL messages are sent directly to WAFL.

Data ONTAP maintains a *default* processing bucket. When the FlexShare service is on, all WAFL messages for a volume that has a FlexShare priority configuration are placed into a dedicated bucket. All WAFL messages associated with volumes that do *not* have a FlexShare priority configuration are placed in the default processing bucket. It is generally a good idea to assign priorities to all active volumes when turning on FlexShare. See the FlexShare technical report for more explanation and details.

VOLUME PRIORITIES

FlexShare affects the order in which outstanding WAFL messages are processed by the storage system. Data ONTAP determines the order based on the FlexShare priority configuration. With FlexShare turned on and configured, higher priority is given to WAFL messages originating from higher-priority volumes.

SYSTEM VERSUS USER OPERATIONS

FlexShare provides a configuration option for *user* versus *system* priority, allowing system-initiated operations to be controlled relative to user-initiated operations. This configuration is available on a per-volume basis.

FlexShare determines whether a WAFL message is a user or a system message based on the origin of the message. If the origin of a WAFL message is a data access protocol, the message is considered to be a user message. All other WAFL messages are considered to be system messages.

The work associated with reading and writing directory metadata blocks and updating inodes uses WAFL *system* messages. Even if the directory is being read due to a user protocol request, the actual WAFL messages used to load directory blocks into memory are considered system activity. Having control over the priority of metadata operations is very useful for tuning high-file-count environments.

CACHE POLICIES

Cache policies are particularly interesting in the context of high file counts. Data ONTAP uses the cache to store buffers in memory for rapid access. When the cache is full and space is required for a new buffer, Data ONTAP uses a modified least recently used (LRU) algorithm to determine which buffers should be discarded from the cache.

FlexShare can modify how the default cache policy behaves by providing hints for the buffers associated with a volume. FlexShare provides hints to Data ONTAP by specifying which information should be kept in the cache and which information should be reused. The cache policy configuration is based on a per-volume setting.

If all volumes hold data that is operated on equally, then the cache policies should either all be kept at the default level or match the volume priorities. The exception is if there are two volumes of equal priority but the administrator knows that one contains data that will never be reused (for example, scan and summarize workloads) or the opposite (for example, reference a small set of files or blocks over and over). In that case, one volume should use a different caching policy over the other.

The cache policies also apply to FlashCache.

5.2 HOW FLEXSHARE HELPS HIGH-FILE-COUNT ENVIRONMENTS

One of the key performance challenges for high-file-count environments is that if the directory metadata is large, it must be completely in memory for many directory operations, and the working set is typically much larger than system memory. This means that memory is a highly contested resource, and the default operation of Data ONTAP may not be managing it optimally for the application.

That's where FlexShare comes in. It allows the administrator to give Data ONTAP hints about how it should prioritize resources differently from the default. All three of the FlexShare tuning knobs may be useful in these environments.

When reacting to unexpected performance problems in a high-file-count environment, FlexShare is one option that can be tried quickly to potentially alleviate problems. FlexShare currently does not need to be licensed and can be turned on or off whenever the administrator desires.

GIVING PREFERENCE TO "SYSTEM" OPERATIONS

WAFL metadata operations are classified as "system" operations, even if they are performed on behalf of user requests. For example, imagine that a user creates a new file in a certain directory. If that directory information is not in memory, all the reads associated with directory metadata are system operations. So are all requests to update the directory metadata, the inode metadata, and so on. The actual file blocks written to the new file are considered "user" operations.

When the working set exceeds memory anyway, we might as well prefer metadata operations. These operations tend to happen much more frequently in high-file-count environments, and directory metadata tends to be referenced more frequently than the actual file data. Giving preference to system operations for high-file-count volumes is one of the most effective ways to improve application-level performance.

GIVING PREFERENCE TO SOME VOLUMES OVER OTHERS

The second FlexShare tuning knob that can help is for giving preference to volumes that hold performance-sensitive data. It is common for a NetApp system to combine volumes for several applications and sets of users on the same system. In addition, high-file-count applications tend to be actively used, their performance very noticeable to users, and their need for system resources very high.

FlexShare can be used to give preference to both system and user operations related to high-file-count volumes. This will probably result in degraded performance for other volumes, but it will improve performance for the high-file-count volumes that need the performance gain.

MARKING LESS CRITICAL VOLUMES FOR REUSE

Finally, FlexShare gives the administrator the ability to mark certain volumes as candidates for giving up room in the system memory (cache) and Flash Cache. The ideal case for high-file-count environments is that all working set directory and inode metadata stays in system memory, as do all the most frequently used file blocks.

Since system memory is limited, the best way to approach that ideal is to mark other volumes for reuse, and to mark the most critical high-file-count volumes to be kept in the system memory. Of course, this may reduce performance for the other volumes, but it is likely to improve performance for the high-file-count volumes that need it most.

Also, some workload patterns do not benefit from system memory, and just "pollute" the cache. These are typically applications that perform some operation on every block of many files, but rarely operate on the

same block twice. For example, imagine calculating the MDA hash for every file in a certain volume. All of the files in the volume are read one at a time, and after the hash is computed they are never touched by the application again. Any blocks associated with already processed files take up space in the system memory that could be better used by other applications.

PUTTING IT ALL TOGETHER

Suppose that you've analyzed the collection of volumes on the NetApp system, understand which volumes are high file count and need greater resources for acceptable performance, and know which volumes are not sensitive to performance. You can now use FlexShare to assign resource usage guidelines to reflect that understanding.

First, mark the performance-sensitive high-file-count volumes as having the FlexShare System and Level settings High or Very High and the Cache setting as Keep. Next, identify volumes that are not performance sensitive, and set their level as Low or Very Low. Identify any noncritical volumes that hold data for applications that don't reference the data, and mark their Cache setting as Reuse. Finally, when using FlexShare it's best to assign explicit settings for all volumes in the system. Any volumes that were not otherwise set should have their FlexShare Level, System, and Cache settings configured as Medium, Medium, and Default, respectively. See TR-3459 for more information and best practices.

In addition, remember that FlexShare is dynamic; you can change the setting on the fly or based on a schedule. For example, if you run nightly jobs on a set of volumes to which you usually assign few resources, you can run a script before the nightly job to increase those system resources and another script to decrease them after the job completes. Consider if and when it is appropriate to dynamically adjust FlexShare priorities based on application or business needs.

5.3 CONFIGURATION EXAMPLE

This section examines a configuration example in depth.

SCENARIO OUTLINE

Suppose that there is some storage for an online multiuser game that is used to help build a virtual world. A couple of key game subsystems use the storage. The most critical subsystem is the map of the virtual world, where each 1,000 square meters of the world (approximately 31 meters, or 100 feet, on each side) are represented by a small JPG picture of the terrain. The world map is currently about 500 kilometers (500,000 meters) on each side, which means that it's represented by approximately 250 million map tiles ($50{,}000 \times 50{,}000$).

The metadata for that many files is very large: about 7.6GB of directory metadata and 44.7GB of inodes. Not all the inodes need to be in memory at the same time, but it is best to have all the directory information in memory.

The system also hosts the catalog of weapons, player types, monsters, and magic spells that show up in this world. Finally, the system also hosts a "dead list" of objects and participants that have been put out of commission in the last 48 hours and need to be cleaned up from the world by a reaper process that runs periodically.

The first two applications are interactive and require decent performance, but although the map is considered "real time," the users can expect to wait a few seconds to retrieve catalog information. The reaper process is a scheduled batch process and is the least sensitive to performance; as long as it can completely scrub the entire virtual world before the next scheduled run, the application functions as designed.

CONFIGURATION STEPS

The first step is to optimize access to the map volume.

```
Fionovar1> priority on
Priority scheduler starting.
Wed Oct 25 22:07:11 GMT [wafl.priority.enable:info]: Priority scheduling
being enabled

Fionovar1> priority set volume maps level=High system=VeryHigh cache=keep
```

We then need to explicitly set priorities for the catalog volume and follow the best practices. However, we know that the catalog is going to be referenced frequently, so we want it to stay in memory if possible.

```
priority set volume catalog level=Medium system=Medium cache=keep
```

Finally, we want to set priorities for the "dead list" volume to be as low as possible. If anybody uses the system at 2 a.m. when the reaper runs, we don't want the reaper to degrade the person's performance more than absolutely necessary.

```
priority set volume reaper level=Low system=Low cache=reuse
```

5.4 FLEXSHARE WRAP-UP

This section provided an overview of FlexShare and how this Data ONTAP feature can help in high-file-count environments. If you need a deeper understanding of FlexShare, see the FlexShare technical report.

6 SIZING FOR HIGH-FILE-COUNT ENVIRONMENTS

Now that you are familiar with the theory and best practices for high-file-count environments, let's look at what it takes to size NetApp systems to support them. The way to size in the general case is beyond the scope of this paper, which focuses on those areas that are the most important for high-file-count environments.

6.1 GATHER REQUIREMENTS

The first step is to gather requirements and to understand what success looks like from the perspective of application performance. This is usually expressed in terms of number of simultaneous users, throughput in transactions or bytes per second, and so one. It's also important to have a specific service level in mind. This is usually expressed in terms of latency in milliseconds per storage request. In large systems, many operations happen at once, and while the combined throughput of those operations can be large, the closer the system is to its utilization limits, the larger the latency.

With these general performance requirements in mind, let's focus on the specifics for high-file-count environments.

6.2 GATHER INFORMATION ABOUT THE APPLICATION ENVIRONMENT

The most relevant and important information to understand is what the key datasets look like. Based on your knowledge or conversations with the application experts, estimate a value for each of the following areas.

TOTAL NUMBER OF FILES

This is the most critical piece of information, and it drives most of the remaining choices. It may be communicated directly (for example, 250 million files total). It may be communicated in terms of users (for example, 1,000 users with an average of 10,000 files per user) and require some math to figure out the total. It may be communicated in some other application-related terms (for example, 200 whirly-jigs

operating on 23,000 cabinets every two weeks) and require deeper application knowledge to figure out the total. Whatever the mode, make your best estimate of the total number of files.

As an illustration, recall the earlier example of the online game. We learned that the "world" was 500km on each side, and that there is a map file for each 1,000 square meters. Doing the arithmetic, we see that the number of files is:

Equation 1) Calculating metadata based on the number of files.

$$\frac{\frac{500km}{1000\frac{m}{km}} \times \frac{500km}{1000\frac{m}{km}}}{1000\frac{m^{\wedge 2}}{file}} = 250{,}000{,}000\,\text{files}$$

Knowing the total number of files and that each name is less than 16 characters long, we can quickly estimate the amount of *on-disk metadata* associated with the file set using Equation 2:

$$MetaDataBytes = 4096\frac{bytes}{dirBlock} \times \left\lceil \frac{NumFiles}{\left\{ 128_{NFS}\frac{files}{dirBlock} \middle| 53_{CIFS}\frac{files}{dirBlock} \right\}} \right\rceil + (192bytes \times NumFiles)$$

Equation 2) Calculating the total number of files based on application parameters.

DIRECTORY ORGANIZATION

This subsection sketches how the files are going to be organized into directories and subdirectories. Are all the files going to be dumped into a single directory? How many top-level directories will there be? For each top-level directory, how many subdirectories will there be, and so on? This information allows more precise estimates of metadata, and also gives you a sense of the pathname lengths typically referenced by the application.

In the game example, suppose that each top-level subdirectory represents a square of territory that is X kilometers on each side. That's a logical way to divide the application data. We now need to figure out what X should be so that we have decent locality of reference without making the directory metadata too large.

It's possible to set up precise equations, but we can just make some quick estimates. The key is to either avoid using indirect directory blocks or, if we use them, to take advantage of the additional space.

If we say that X equals 1, that means we will have 250,000 top-level subdirectories. At 128 subdirectories per 4KB directory block, that's about 8MB of directory metadata. That's not much by the standards of modern storage platforms, and since this is the set of top-level subdirectories, we can be sure that the directory is likely to reside in memory all the time.

Each map tile represents 1,000 square meters and each subdirectory represents a square kilometer. This means that each top-level subdirectory needs to hold 1,000 files. For a 32-bit flexible volume, that can be accomplished without using any indirect directory blocks and represents just 32KB of directory metadata. It also allows the application to address all of the map tiles with just two levels of subdirectories. This is useful, because each directory level represents a separate name resolution over NFS.

Now try a similar exercise with different levels of X, but also set yourself the constraint of never using any indirect directory blocks. It's an interesting exercise, and may be useful in other situations in which the

working set is very large and the disk subsystem is slow, thereby penalizing the use of indirect blocks for directories that don't stay in memory very long. The down side with deeper directories is that each subdirectory name resolution is done separately for NFS.

AVERAGE FILE NAME LENGTH

Now that you know how the directories will be organized, take note of how the directories and files will be named. If the names are long, they may require additional directory metadata to hold all the name chunks. In addition, if all file names are very long, they may not be able to fit in the name cache and require resolution for every reference.

File and directory names for high-file-count applications are usually very logically constructed. They are typically things like timestamps, hash code values, coordinates in some application space, and so on. If possible, keep those values short.

For example, if the file or directory name is a timestamp, it's possible to encode it as an ASCII string that spells out the date and time: "`24October2006@18:05:30`". That uses 22 characters and would use up at least two name chunks in the WAFL directory block because it's longer than 16 characters. An alternative would be to encode the date as the number of seconds past a certain time in hexadecimal. That way, it's possible to encode over 100 years using just 8 characters. It's then a matter of using some common routines to convert to and from alternative timestamp formats.

This approach not only reduces the amount of directory metadata needed, it also reduces the pathname lengths and helps improve name cache performance.

Building on the previous example, we can represent each top-level directory with a short name that represents the coordinates of that area on the map (for example, `107x352`) and each file by a similar name (`07x23`). That would make the path to each file relatively short (for example, `/vol/map/107x352/07x23`) at 22 characters for every file in the map.

IS IT AN NFS-ONLY ENVIRONMENT?

As mentioned previously, an NFS-only environment allows each file name that is 16 or fewer characters to use a single name chunk in the directory block. If the environment is CIFS only or mixed, then WAFL will use 3 name chunks per 16 characters of file-name length.

As a reminder, there are currently 128 name chunks per 4KB directory block. In an NFS-only environment, you can estimate 4KB of directory information per 128 files

$(DirMDsize = 4KB \times \lceil NumFiles/128 \rceil)$.

Otherwise, estimate 4KB per 53 files

$(DirMDsize = 4KB \times \lceil NumFiles/53 \rceil)$.

Additional metadata is associated with indirect blocks used to hold directory information, but that tends to be orders of magnitude smaller than the actual directory blocks. For a first-cut estimate, it's best to ignore them and simplify the calculations. If it makes sense, you can go back later and compute the exact sizes based on the theory discussed previously.

AVERAGE FILE SIZE

How large are the files being stored on the system? In typical high-file-count environments, the actual files tend to be small. However, the file capacity adds up and is the primary indicator of the minimum disk capacity needed to hold the dataset. This can be expressed as an overall average (for example, 12KB), or as some mix of sizes that can be used to compute a weighted average (for example, 50% 1KB plus 25% 10KB plus 25% 100KB equals 28KB average).

Average file size is sometimes communicated indirectly. If you know that the total size of the 2 million files adds up to 14GB, you can calculate that the average file size is about 7,516 bytes.

WORKING SET

This is where it gets much harder to just ask the application expert. It's very unusual for an application to already be in operation and sufficiently instrumented to get an accurate sense of its working set. Our most likely approach is to approximate the working set by looking at the amount of storage expected to be accessed at the same time. This expectation should be for the peak time at which we need to deliver acceptable performance. This expected amount of storage is derived from two dimensions.

The first dimension is the *number of directory blocks being accessed simultaneously.* All of the directory blocks that need to be accessed simultaneously should be able to fit in memory. If they don't fit, and the directories are large, application response times are degraded significantly while the directory blocks are loaded into memory (and possibly force out other blocks that are part of the working set for another process).

The second dimension is the *number of files being accessed simultaneously.* This determines both the number of inodes that need to be cached and the number of file data blocks that need to be in memory.

The amount of memory used per file (inode) and per directory block is not the same as the amount of space that data uses on disk. Data ONTAP builds and maintains auxiliary data structures in memory to provide efficient and fast operation. Those additional data structures also use memory. For directories, Data ONTAP maintains hash tables to accelerate directory lookup operations. *These hash tables are approximately 20% of the actual directory size*. So if the directory used 100MB on disk, an additional 20MB of memory would be required to store the hash table. Also, for each file (inode) in memory, Data ONTAP maintains approximately 500 bytes of metadata in addition to the inode. That works out to approximately 700 bytes of system memory required for each file in the working set. For example, a working set of 10 million files would require approximately 6.5GB of system memory to manage the file metadata in addition to any memory used to store and manage the directory information and actual file contents.

By estimating the working set size in terms of bytes, it's possible to predict whether a particular storage platform will *not* work. For example, if the estimated working set is 37GB and the platform being considered has only 8GB of controller memory, then we know that platform will definitely *not* work.

It's harder to say for sure that a working set close to the controller memory size will work. For example, if the working set is 30GB and the controller memory is 32GB, it's not clear that performance will be good. There may be additional processes competing for controller memory, the estimate could be inaccurate, and so on. Any part of the working set that is not in controller memory must be fetched from disk. Since the references to disk take several orders of magnitude longer than memory references, a working set that is even a little too large for memory rapidly causes performance to degrade.

Alternatively, it's possible to work backward from the controller memory to estimate the largest possible working set that the controller can support. If the application requires a larger working set, then either a larger controller or more controllers are needed.

6.3 CALCULATE ADDITIONAL RESOURCES REQUIRED

Now that we have an approximation of the application needs, we can calculate any additional space that is needed on disk and in memory.

You can use the information in "Background and Theory of Operation," earlier in this report, to estimate the size of directory metadata. In most cases it's not necessary to take the indirect blocks into account when calculating the directory metadata—for high-file-count environments, they are a very small percentage of the overall metadata.

METADATA SIZE FOR DIRECTORY STRUCTURES

When calculating directory metadata, the key is to remember that each directory block uses 4KB of storage. That block can reference up to 128 files (NFS only) or 53 files (mixed or CIFS only). For example, if the application requires 100 million NFS-only files, divide that number by 128 and then multiply the result by 4KB to estimate the size of directory metadata in kilobytes. Convert to megabytes or gigabytes for convenience. In this example, the directory metadata requires about 3GB. See Equation 1,

earlier in this document, for details. If file names average more than 16 characters, then you need to go back and more precisely estimate how many file references can be held in each directory block. To calculate the amount of memory required to operate on the directories, add another 20% to account for the directory hash tables.

INODE FILE SIZE FOR TOTAL NUMBER OF FILES

In addition to the directory metadata, each file and directory must also be referenced by an inode. As stated previously, an inode currently uses 192 bytes. So for each file and directory used by the application, add 192 bytes to the required metadata. For example, assume that the 10 million files mentioned earlier are distributed among 10,000 directories. Multiply the inode size (192 bytes) by 10,010,000 to obtain the amount of required inode storage in bytes. This number can be converted to megabytes or gigabytes for convenience. In this example, the inode information would require over 1.8GB on disk.

Not all the inodes for the volume need to be in memory at the same time; the inode file size exceeding physical memory is not an issue by itself. Only in cases in which the working set of files does not fit in physical memory should you consider a larger controller or a more distributed architecture. It's very important to remember that while an inode takes up 192 bytes on disk, when the associated file is loaded into memory, additional metadata is used to a total of approximately 700 bytes per file. For our example in the previous paragraph, the total amount of system memory required would be approximately 6.5GB.

ADD TO TOTAL CAPACITY OF APPLICATION DATA BEING STORED

Once you have estimated the total additional metadata required by the application, add it to the storage required to hold the file data. For example, suppose that you expect the application to store 200GB of file information (an average of about 2KB per file). Add 3GB for directory metadata and another 1.8GB of inode data. That raises the total amount of storage required to 205GB.

6.4 CHOOSE THE BEST SOLUTION ARCHITECTURE

Now that we understand the application requirements and additional resources required to accommodate the WAFL metadata, we can more easily choose a storage platform and overall architecture to meet the needs of the application.

CONTROLLER MEMORY SIZE

The most important factor is the amount of physical memory in the storage controllers. The controller memory must hold as much of the application working set as possible. At the minimum, it must hold:

- Data ONTAP processes
- The directory
- Inode and other per-file metadata associated with the working set
- Enough of the working set file data so that the disk subsystem can supply the rest with low enough latency to meet application service-level requirements

TOO LARGE FOR A SINGLE CONTROLLER

If it turns out that no single controller can accommodate the entire working set of the application, then multiple controllers must be used and the application data divided among them.

For example, suppose that after careful calculation you conclude that the application requires a working set of 100GB and the largest available controller holds only 96GB. Then the storage solution must be designed to divide the working set among two or more controllers.

Suppose that we use two controllers with 96GB of memory each. This allows enough space to accommodate the entire working set with room to grow. However, we must now make sure that the application can access its storage across two physically separate Data ONTAP controllers. There are a

number of ways to accomplish this scaling; however, describing them in depth is beyond the scope of this paper.

CLUSTER-MODE

One of the key features of Data ONTAP operating in Cluster-Mode is the ability to provide a global namespace within a cluster. To provide this capability, a new "junction" file type is added to WAFL. A junction serves as a redirection point that acts similarly to a symlink to a different volume.

If the working set of an application exceeds the limitations of a single physical controller, the customer can scale out by deploying Data ONTAP operating in Cluster-Mode. By scaling out, additional controllers can be added seamlessly to a pre-existing resource pool and clients can now access a very large data container using a single mount point. The name space consists of many volumes stitched together under a single mount point. The client just sees volumes as directories and has no knowledge of volume boundaries. Please reference TR 3982 Data ONTAP 8.1 Operating in Cluster-Mode: An Introduction for more details.

DISK SUBSYSTEM

While controller memory is the most important factor, you must also take the disk subsystem into consideration. There are two key dimensions: usable capacity and access latency.

USABLE CAPACITY

Disk storage is often described in terms of raw disk capacity, usually expressed in terms of billions (10^9) of bytes of available storage on the disk drive. It's a somewhat confusing marketing practice, since application storage requirements are often stated in terms of gigabytes (2^30). That translation between powers of 10 and powers of 2 loses about 7%. For example, 500 billion bytes translates to 465GB.

From that point, WAFL "right-sizes" the disks to more efficiently work across a range of manufacturers and drive sizes. This gives some additional space, since we end up taking the lowest common denominator. We also use part of the block on disk to hold error correction information for higher reliability.

After right-sizing each disk, WAFL organizes the disks into RAID groups with parity drives to provide protection against drive failures. Drives holding parity information can't hold application data, and so are subtracted from usable capacity.

In addition, WAFL by default reserves 10% of remaining space to give itself room for the "write anywhere" operations, which require a certain amount of free space to work efficiently.

Finally, one of the great features of WAFL and Data ONTAP is the efficient Snapshot™ technology capability. By default, Data ONTAP assigns 5% of the remaining free space to be set aside for data and metadata held in Snapshot copies (releases prior to Data ONTAP 8.0 have a default Snapshot copy reserve of 20%).

RESPONSE TIME (LATENCY)

Another important factor is the capability of the disk subsystem to serve requests for data on disk at an acceptable response time. The response time (latency) is measured in milliseconds, and is related to the number of revolutions per minute (RPM) of the disks used in the aggregate. The higher the RPM, the lower the average response time per request for a given request load. Each disk is capable of serving a fixed number of requests per second at a given response time limit. If we know the response times required by the application, we can calculate the minimum number of data spindles required to meet those needs. More information about the disk subsystem and how to perform those calculations is available from the NetApp internal document TR-3437: Storage Best Practices and Resiliency Guide.

DIRECTORY LAYOUT

We've already discussed directory layout at length, so be sure to take the various factors into account when recommending or evaluating a directory layout. In many situations, there tends to be little control over directory layout. The application expects directories to be arranged a certain way. Or there is already

a large set of data in a certain layout, and it's not practical to rearrange it. If you are in a situation in which you can guide or recommend a directory layout for a high-file-count environment, here are a few factors to keep in mind:

- Most directory layout choices have approximately the same metadata, since the bulk of the metadata is associated with the files rather than with the subdirectories. For example, a single layer of indirect directory blocks uses 8 indirect blocks to reference up to 4,080 directory metadata blocks for a 32-bit flexible volume. In a 64-bit flexible volume, a single layer on indirect directory blocks uses 5 indirect blocks to reference up to 1,275 directory metadata blocks.
- Remember the indirection transition points. For example, for 32-bit flexible volumes, for CIFS, directories with more than 422 files need to use indirect blocks, and those with more than 216,238 files need to use double-indirect blocks. For NFS, directories with more than 1,022 files need to use indirect blocks and those with more than 522,238 files need to use double-indirect blocks. For a 64-bit volume, for CIFS, directories with more than 263 files need to use indirect blocks, and those with more than 67,573 files need to use double-indirect blocks. For NFS, directories with more than 638 files need to use indirect blocks and those with more than 163,198 files need to use double-indirect blocks. It's a good idea to avoid using double-indirect blocks for more than a few directories that are expected to consistently stay in cache.
- Try to rearrange the directory layout to avoid making rarely used directories very large. When that directory is accessed, it will push out a large amount of more frequently referenced information from the cache.
- If a large directory is used frequently, make sure the metadata can fit entirely in system memory.
- It is better to have shorter file paths and shallower directory trees. This results in fewer subdirectory name resolution requests and better pathname caching. If you need to choose between wider branching toward the top of the tree and having deeper directory trees, choose wider branching. Most modern platforms have the memory to cache the wide directories. If you are forced to use a hardware platform with a smaller amount of memory, you may need to use smaller directories and deeper directory trees, and hope that the working set fits within memory.

SYSTEM TUNINGS

In addition to choosing the appropriate platform, you may need to adjust the various Data ONTAP tunable parameters mentioned previously:

- `maxdirsize`
- `maximum inode count`
- `no_atime_update`
- `minra`
- FlexShare

Running Reallocate in a high-file-count environment will spread the data across all disks evenly. However, it typically will not help performance, since high-file-count environments are usually random in nature.

If you skipped to this section, see the "Key Best Practices to Remember" section later in this report for additional discussion.

7 QUICK REFERENCE

Table 1) WAFL limits.

WAFL structure	Limit
Size of on-disk inode	192 bytes
Size of in-core inode	~700 bytes
Size of inode file	192 bytes * Total number of inodes
Maximum inode count	MINIMUM of (2 billion, 1 inode per 4Kb of volume size)
Maximum number of subdirectories in a directory	Data ONTAP 8.1: No specific limit Pre-Data ONTAP 8.1: 99,998
Maximum number of entries in a directory	Depends on the `maxdirsize` setting

Table 2) Directory information.

Directory object	Size
Size of a directory block	4,096 bytes
Number of directory entries in a block	128
Number of name chunks in a block	160
Name chunks required for a 16-byte NFS file	1
Name chunks required for a 16-byte CIFS file	3

Table 3) Various values for 32-bit flexible volume, 64-bit flexible volume, and traditional volume.

	32-Bit Flexible Volume	64-Bit Flexible Volume	Traditional Volume
Block pointer size	32	64	32
Number of block pointers in inode	8	5	16
Number of block pointers in an indirect block	510	255	1,024
Maximum number of NFS directory entries with a single layer of indirection (assuming 16-byte file names and 128 entries per directory block)	128 * 8 = 1,024	128 * 5 = 640	128 * 16 = 2,048
Maximum number of CIFS directory entries with a single layer of indirection (assuming 16-byte file names and 53 entries per directory block)	53 * 8 =424	53 * 5 = 265	53 * 16= 848
Maximum number of NFS directory entries with a double layer of indirection (assuming 16-byte file names and 128 entries per directory block)	128 * 510 * 8 = 522,240	128 * 255 * 5 = 163,200	128 * 1024 * 16 = 2,097,152
Maximum number of CIFS directory entries with a double layer of indirection (assuming 16-byte file names and 53 entries per directory block)	53 * 510 * 8 = 216,240	53 * 255 * 5 = 67,575	53 * 1024 * 16 = 868,352

Table 4) Default Maxdirsize values for various platforms.

Platform	Maxdirsize Setting (KB) (default 1% of system memory)	Number of NFS Entries	Number of CIFS Entries
FAS2040	41,943	1,342,176	555,744
FAS2240	45,875	1,468,000	607,843
FAS3040	41,861	1,339,552	554,658
FAS3070	83,804	2,681,728	1,110,403
FAS3140	41,861	1,339,552	554,658
FAS3170	167,690	5,366,080	2,221,892
FAS3210	45,875	1,468,000	607,843
FAS3240	73,400	2,348,800	972,550
FAS3270	190,054	6,081,728	2,518,215
FAS6030	167,690	5,366,080	2,221,892
FAS6040	167,690	5,366,080	2,221,892
FAS6070	335,462	10,734,784	4,444,871
FAS6080	335,462	10,734,784	4,444,871
FAS6210	251,658	8,053,056	3,334,468
FAS6240	327,680	10,485,760	4,341,760
FAS6280	327,680	10,485,760	4,341,760

Table 5) Commands.

Command	7-Mode	Cluster-Mode
Set maxdirsize	`vol options <vol> maxdirsize`	`vol modify –volume <vol> -maxdir-size`
View maxdirsize	`vol status –v <vol>`	`vol show –volume <vol> -field maxdir-size`
Set minra	`vol options <vol> minra`	`vol modify –volume <vol> -min-readahead`
View minra setting	`vol status –v <vol>`	`vol show –volume <vol> -field min-readahead`
Set create_ucode	`vol options <vol> create_ucode`	Not supported
View create_ucode	`vol status –v <vol>`	Nodeshell cli: `vol status –v <vol>`
Set convert_ucode	`vol options <vol> convert_ucode`	`vol modify –volume <vol> -convert-ucode`
View convert_ucode	`vol status –v <vol>`	`vol show –volume <vol> -field convert-ucode`
Set no_atime_update	`vol options <vol> no_atime_update`	`vol modify –volume <vol> -atime-update`
View no_atime_update setting	`vol status –v <vol>`	`vol show –volume <vol> -field atime-update`

View inode counts	`df -i`	`vol show -volume <vol> -field files`
		
		`vol show -volume <vol> -field files-used`

Change maximum inode count	`maxfiles`	`vol modify -volume <vol> -field files`
View current range of inode allocation (post-Data ONTAP 8.0 only)	`maxfiles -capacity`	`vol show -volume <vol> -field inodefile-public-capacity`

8 SUMMARY

We hope that this exploration into high-file-count environments has been helpful for understanding and resolving the challenges involved. In this technical report, we have:

- Delved into the theory of WAFL file layout and directory design
- Reviewed applicable best practices
- Learned a methodology for sizing high-file-count environments

8.1 KEY BEST PRACTICES TO REMEMBER

As you go forward using what you have learned, keep these best practices in mind.

Best Practices

- Platform memory is a critical factor for high-file-count environments. Choose the platform with the most memory possible, and if the metadata plus working set is larger than physical memory, split the work across several controllers.
- Keep the directory trees bushy (relatively flat with nontrivial branching at each level). At the leaf subdirectories, try to limit the average number of files to fewer than 800 (CIFS and mixed access) or 2,000 (NFS only).
- Try to keep file names shorter than 16 characters. If longer names can't be avoided, most of your file names need to be shorter than 40 characters (ASCII) or 20 characters (CIFS/Unicode) for the name cache to work efficiently.
- Use higher-RPM disks and more spindles to reduce access latency when going to disk for the data.
- Use RAID-DP exclusively to avoid a large performance impact on disk reconstruction.
- Keep plenty of free space in volumes and aggregates to give WAFL lots of working space to optimize layout.
- Delete directories that once held thousands or millions of temporary files and move any permanent files to another directory.
- Use FlexShare when possible to optimize access to key volumes and metadata.
- Set WAFL-tunable parameters to accommodate the required file set and directory sizes.
- Expect NDMP and QSM to take a very long time to transfer all the files, and plan to use block-based backup mechanisms (for example, volume SnapMirror and SnapMirror to tape).
- Deploy Flash Cache.

8.2 WHERE TO GO FOR MORE HELP

If the information in this technical report is not sufficient, please contact NetApp technical experts.

NetApp provides no representations or warranties regarding the accuracy, reliability, or serviceability of any information or recommendations provided in this publication, or with respect to any results that may be obtained by the use of the information or observance of any recommendations provided herein. The information in this document is distributed AS IS, and the use of this information or the implementation of any recommendations or techniques herein is a customer's responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. This document and the information contained herein may be used solely in connection with the NetApp products discussed in this document.

Go further, faster®

www.netapp.com