NetApp Technical Report

FlexCache Caching Architecture

Marty Turner, NetApp
July 2009 | TR-3669

STORAGE CACHING ON NETAPP SYSTEMS

This technical report provides an in-depth analysis of the FlexCache™ caching architecture. You will learn how FlexCache works, how data is stored in its cache, how it manages cache consistency, and how storage space is used within the NetApp® system.

For an introduction to FlexCache, the business problems it can solve, and use cases, read the FlexCache datasheets available online.

TABLE OF CONTENTS

1 PURPOSE AND AUDIENCE

This technical report provides an in-depth analysis of the FlexCache caching architecture. You will learn what FlexCache stores in its cache; how it manages cache consistency; the process for hits, misses, and internal communications flow; and how storage space is used within the NetApp system.

This technical report is intended for storage administrators, evaluators, and implementers who want to learn more about FlexCache than can be learned from the information provided in the "FlexCache Design and Implementation Guide." After reading this report, you will have a much better understanding of how FlexCache works.

2 INTRODUCTION

FlexCache is a caching technology that provides a cache architecture at the storage protocol layer. Similar to the way a cache in the memory architecture of a compute system improves performance, FlexCache improves performance in your NFS environments by scaling out cache volumes for increased IOPs, bringing data closer to your hosts for decreased latencies, off-loading overburdened storage controllers, or a combination of all of these.

TERMINOLOGY USED IN THIS TECHNICAL REPORT

- **Host.** A host, application, or client that requests data from networked storage. Specific to FlexCache, hosts are always mounted to a FlexCache volume through the NFS v2 or v3 protocol.
- **Cache volume.** A Data ONTAP® volume that is created as a FlexCache caching volume.
- **Origin volume.** A Data ONTAP volume that is mapped to a FlexCache caching volume as the original source of data.

2.1 CACHING

A cache is a temporary storage location that resides between a host and a source of data. The main objective of a cache is to store frequently accessed portions of a source of data in a way that allows the data to be served faster and/or more efficiently than it would be by fetching the data from the source. Caches are beneficial in read-intensive environments in which data is accessed more than once and/or is shared by multiple hosts.

A cache can serve data faster in one of two ways:

- The cache system is faster than the system with the data source. This can be achieved through faster storage in the cache (for example, FC versus SATA), increased processing power in the cache, and increased (or faster) memory in the cache.
- The storage space for the cache is physically closer to the host, so it does not take as long to reach the data.

Caches are implemented with different architectures, policies, and semantics so that the integrity of the data is protected as it is stored in the cache and served to the host. The following sections describe the way FlexCache implements its cache architecture.

2.2 READS

Caches are populated as a host reads data from the source. On the first read of any data (1 in Figure 1), the cache has to fetch the data from the original source (2). The data is returned to the cache (3), stored in the cache, and then passed back to the host (4). As reads are passed through a cache, the cache fills up by storing the requested data.

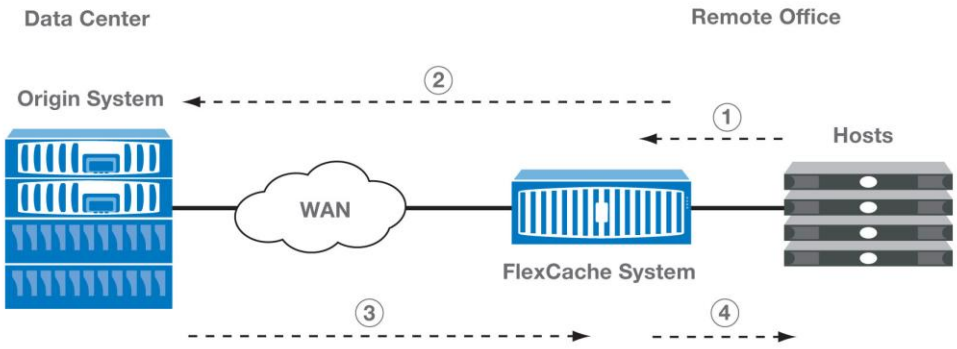

Figure 1) First read of data.

Any subsequent accessing of data (1 in Figure 2) that is already stored in the cache can be served immediately back to the host (2) without spending time and resources accessing the original source of the data. This is the primary advantage of a cache—serving frequently accessed data directly to a host without having to fetch the data from the original source. However, you may be thinking, "What if the data changed on the origin system? Does the FlexCache system still serve the data stored in its cache?" It is possible that the FlexCache system could be storing data that has changed at the origin system—this is called *stale data*. However, although it is true that the FlexCache system may store stale data, policies exist that allow you to control and manage how the FlexCache system handles stale data. These policies are discussed in section 3, "Cache Consistency."

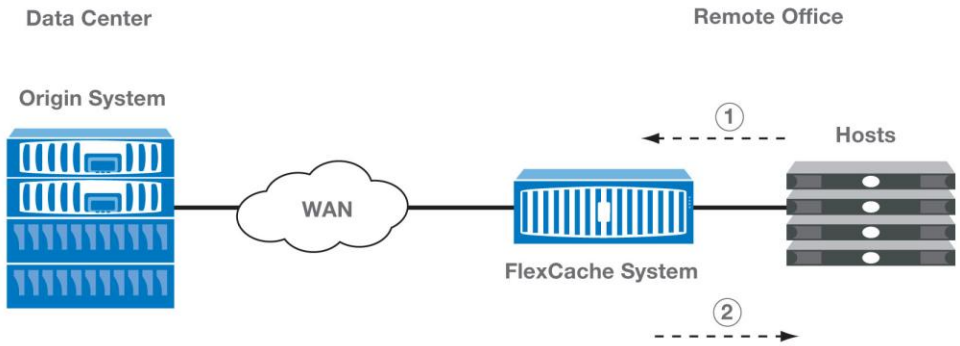

Figure 2) Subsequent read of the same data.

2.3 WRITES

In a FlexCache system, all writes from a host (1 in Figure 3) are passed directly through the cache volume to the origin volume (2). The origin volume responds to the FlexCache volume when it assumes responsibility for the new or changed data (3); only then does the FlexCache volume acknowledge the result of the write to the host (4). This is called a *write-through cache*.

A write-through cache is a cache that does not respond to the host until it receives a response from the next subsystem in the line. In other words, the FlexCache cache volume does not respond to the host until the origin volume acknowledges receipt of the data, thus helping to keep the data safe and sound.

This is in contrast to a *write-back cache*, which responds to the host immediately before verifying that the data can be successfully passed to the next subsystem. Once a write-back cache accepts responsibility for data and responds to the host before acknowledging receipt of the next subsystem, the write-back cache must protect the data until it is written to physical media (i.e., disk). Data in this state is called *dirty data*, and it must be protected from system failures such as power loss (in such a way that when power is restored, the dirty data is still accessible and ready to be stored on disk).

A FlexCache system is not a write-back cache, and therefore it does not store dirty data. Since dirty data is never stored in a cache volume, it is *not* imperative to protect a FlexCache system from power failure. A power failure does not result in data loss or data corruption. A power failure results only in the loss of the host's NFS mount point. One other side effect of restarting a FlexCache system is covered in Section 6.1: Losing the Connection.

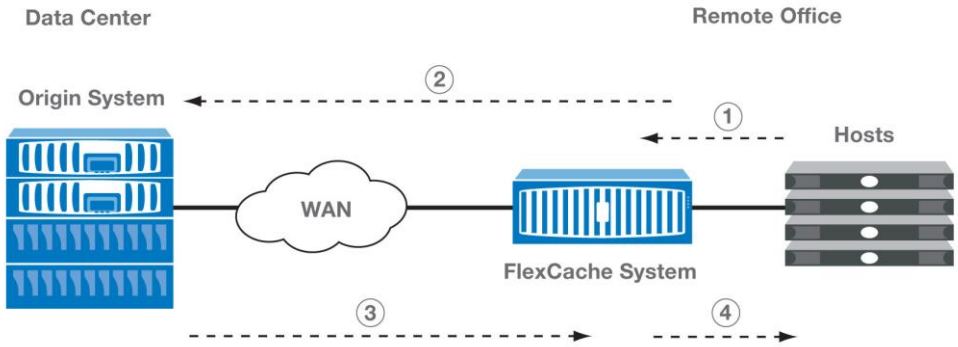

Figure 3) Host writes data to a write-through FlexCache system.

2.4 CACHING GRANULARITY

Over time, a cache volume becomes a collection of bits and pieces from various files, various directories, and various other objects. The collection of data in a cache volume is never intended to represent a fully functional set of the data at any particular point in time. This is called a *sparse volume*. A sparse volume occurs because applications and operating systems request only the portions of data needed at that moment in time; that is, an entire file is not read from the origin for every file access.

There are several types of objects that can be read from a host. A cache volume caches files, directories, and symbolic links. Each object is stored at 4K-block-level granularity. In other words, if a file on the origin volume is 400K but the host requests only the first 8,000 bytes of the file, then FlexCache requests and stores only the first two 4K blocks of the file. This maximizes disk space efficiency. Also, for every file that contains data blocks in the cache volume, the file attributes are cached (meaning that the entire file is considered to be cached). If a file or piece of data that does not exist in the cache is requested from the cache volume, that data is retrieved from the origin volume. Data continues to be stored in the cache volume until the cache volume or its aggregate runs out of space. Section 7 goes into detail regarding space management and what happens when you run out of space.

When writes come from the host for data that is in the cache volume, they must be sent through to the origin volume. If the new data is part of a file that is in the cache, then the entire file is invalidated, meaning that every block of data stored in the cache for that file is no longer used. This can result in invalidating more data than necessary; for example, if you changed only one block of a file but the cache stored 10 blocks of the file, none of the 10 blocks will be used for future reads. Because of this design point, data sets consisting of large files that are frequently updated may not be good candidates for a FlexCache implementation. Also note that the newly written data is not stored in the cache volume; data is stored in the cache volume only as a result of a read through the cache.

While writes to files and directories clearly invalidate the cached object as described earlier, changing the permissions of a file or directory is not as obvious. When the permissions of a file are changed from the host, the file is not invalidated in the cache volume. However, if the permissions of a directory are changed, then the directory object is invalidated in the cache volume.

3 CACHE CONSISTENCY

Cache consistency is the process by which the cache knows how and when to store, invalidate, and serve data. Since data in the cache is, by definition, not the original source of the data, before serving data from a cache you must be aware of the “freshness” or “staleness” of the data, as defined in Table 1.

Table 1) Fresh versus stale data.

Fresh	Stale
The data in the cache is guaranteed to be the same as the data on the origin system.	The data in the cache is definitely *not* the same as the data on the origin system.

EXAMPLE OF FRESH DATA

On the first read of data from a host, the data is not in the cache volume. The cache volume requests the data from the origin volume and immediately stores the data in the cache volume. The attributes of the file are stored with the data (including date and time stamp). This data is *fresh*. It was just retrieved from the origin volume as the latest version of that data.

EXAMPLE OF STALE DATA

Host reads have resulted in data being stored on the cache volume. A host mounted directly to the origin volume changes data in a file that is also on the cache volume. The file on the origin volume can be changed and does not directly update the data on the cache volume. In this scenario, the data on the cache volume is not the latest version of data that is on the origin volume, and is defined as *stale* data.

Three primary policies govern the freshness of data: attribute cache time-outs, delegations, and the write operation proxy. Attribute cache time-outs can be tuned as described in section 3.1; however, delegations and the write operation proxy are governed by the FlexCache implementation (and thus cannot be tuned or disabled). Tuning the FlexCache cache consistency parameters involves balancing between performance and the freshness of data being returned to hosts. Based on your specific data usage, you may be willing to sacrifice data freshness for better performance.

3.1 ATTRIBUTE CACHE TIME-OUTS

As data is retrieved from an origin volume and stored in the cache volume, the file containing that data is considered fresh for a specified amount of time, called the *attribute cache time-out*. In other words, if a host requests data from a file in the cache volume and the attribute cache time-out has not expired, the cache volume serves the data directly to the host without having to communicate with the origin volume. If the attribute cache time-out has expired, the cache volume checks with the origin volume to compare the file’s attributes. If the attributes are the same on the cache volume and the origin volume, the file is fresh, and the data is served from the cache volume (and the attribute cache time-out restarts). If the attributes are not the same, the file is stale, is marked invalid, and is reread from the origin volume before serving the host (also resetting the attribute cache time-out).

A situation may arise in which a file is changed on the origin volume after that file was delivered to a cache volume. If that file on the cache volume is subsequently requested before the attribute cache time-out expires, then the data is served (unknowingly) as stale. If serving stale data in this manner is unacceptable in your environment, you can set the attribute cache time-out to zero to avoid this scenario. With the attribute cache time-out set to zero, data is served with the latest version of the data; however, performance is negatively affected because every request for data results in checking file attributes at the origin volume. This is a trade-off you have to consider based on the usage characteristics for your data.

For many workloads, the working data set does not change at all. An example is rendering applications in which data is read from one file and the contents of that file do not change. After processing on the file, writes are made to another file. Caching, and effectively using attribute cache time-outs, can result in significant performance advantages in such applications.

The default attribute cache time-out is 30 seconds and can be individually tuned, higher or lower, for specific types of objects (file, directories, and symbolic links).

3.2 DELEGATIONS

A delegation is another mechanism used to manage cache consistency. A delegation is a contract between the cache volume and the origin volume, which says that if the cache volume is granted a delegation for a specific file, the origin volume does not change that file without first notifying the cache volume. In other words, if the cache volume holds a delegation for a file, the cache volume is guaranteed that the file is fresh (the latest copy of the file), including the file's attributes. This means that the cache volume does not have to validate the file with the origin volume, even if the attribute cache time-out has expired. In fact, if a file has a delegation, the attribute cache time-out is ignored.

Since having delegations for files in the cache volume results in the most efficient process for serving reads to hosts, let's look at how delegations are granted. When the cache volume requests a fresh copy of data from the origin volume to handle a host read request, FlexCache automatically requests a delegation when appropriate. Delegations are requested only for file data, as opposed to other cached objects (such as directory data and symbolic link data). Some other environments in which delegations are not used include those in which the origin volume is a GX HPO/striped volume or a SnapMirror® volume.

Once delegations are granted to cache volumes from the origin volume, the origin volume is responsible for revoking those delegations if a file changes. When a cache volume is notified of a revoked delegation, it knows to reread the data for the file from the origin volume on the next access from a host. However, if the attribute cache time-out has not expired, the data is still served to the host.

In an environment in which one origin volume is mapped to multiple cache volumes, the origin volume can grant delegations for the same file to one or more of the cache volumes. In this case, if the file is changed on the origin volume, all outstanding delegations for that file are revoked.

3.3 WRITE OPERATION PROXY

The final cache consistency policy is the write operation proxy. If a host modifies a file, the consistency of that file must be considered throughout all the caching and origin systems. This involves three types of systems: (1) The origin volume that contains the file being changed, (2) the cache volume from which the request comes, and (3) any other cache volumes that are mapped to the same origin volume.

For (1), the origin volume that contains the file being changed, the original source file is updated with the host's modifications of the data. This operation is executed when the cache volume proxies the write operation through to the origin volume. Also, the origin volume changes the attributes of the file and revokes delegations from any cache volume holding a delegation on this file.

For (2), the cache volume from which the request comes, if the changing file happens to exist in the cache volume, the file is marked as invalid and is no longer used. On subsequent accesses of the data, the cache volume requests the new data from the origin volume. Also, if this cache volume holds a delegation for this file, the delegation is revoked.

For (3), any *other* cache volumes that are mapped to the same origin volume, the volumes are not notified of the specific file change; however, any outstanding delegations for this file are revoked. If the changing file exists in these *other* cache volumes, since the delegation is revoked, the only time the data will be used is if it is accessed from a host prior to the expiration of the attribute cache time-out. Otherwise, the cache volume verifies the attributes with the origin volume and realizes that the file has been updated.

4 HITS AND MISSES

The lifecycle of data in a cache volume is very dependent on your workload characteristics. Some data lives in the cache volume for a long time; other data becomes stale in a short amount of time; and some data may not even be stored in the cache volume. To determine if the cache volume is operating efficiently, you need to understand two scenarios: (1) the data a host requests can be served from the cache volume, and (2) the data a host requests cannot be served from the cache volume.

4.1 HITS

When the data a host requests is in the cache volume and served directly from it, it is called a *cache hit*. Cache hits are the result of a previous request, because data cannot be served from the cache until it is read the first time from the origin volume and stored in the cache volume. Hits are processed in one of two ways:

- **Hit.** The requested data is in the cache volume and does not need to be verified with the origin volume. This means that a delegation exists for this data; or, if no delegation exists, the attribute cache time-out for the data has not expired. This type of request is served locally and access to the origin system is not necessary. This is the fastest, most efficient way to serve data from a cache volume.
- **Hit-Verify.** The requested data is in the cache volume and verification with the origin volume is required before serving the data to the host. This means that a delegation does not exist for this data and that the attribute cache time-out has expired. The cache volume communicates with the origin volume and determines that the attributes of the data are consistent (the data has not changed on the origin volume since the last time the cache volume accessed it). At this point, the cache volume can serve the data locally from its cache volume and does not have to reread the data from the origin volume. Communication between the cache volume and the origin volume is limited to checking attributes for the file containing this data.

4.2 MISSES

When the data that a host requests cannot be served directly from the cache volume, this is called a *cache miss*. Cache misses can result from one of two scenarios:

- **Miss.** The requested data is not in the cache volume. The data is read from the origin volume and stored in the cache volume for a possible subsequent read.
- **Miss-Verify.** The requested data is in the cache volume and verification with the origin volume is required before serving the data to the host. However, contrary to Hit-Verify, communicating with the origin volume reports that the attributes of the data are not consistent (the data has changed on the origin volume since the last time the cache volume accessed it). The cache volume invalidates the data currently stored in the cache, rereads the data from the origin volume, and stores the new data in the cache volume for a possible subsequent read.

5 PROCESS FLOW CHARTS

Putting everything together, the following flow charts show the process for a read and a write from the host, taking into account:

- **Cache consistency.** Cache attribute time-outs, delegations, and write operation proxy
- **Hits.** the Hit and the Hit-Verify
- **Misses.** the Miss and the Miss-Verify

5.1 READ COMMAND

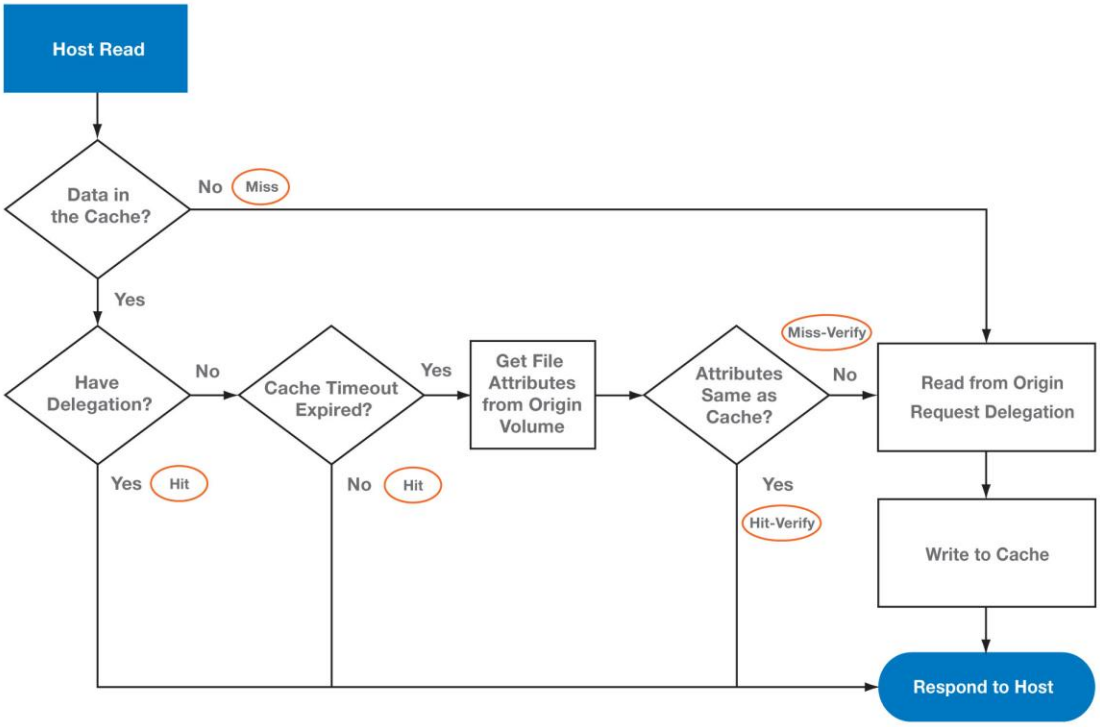

Figure 4) Read from a host.

5.2 WRITE COMMAND

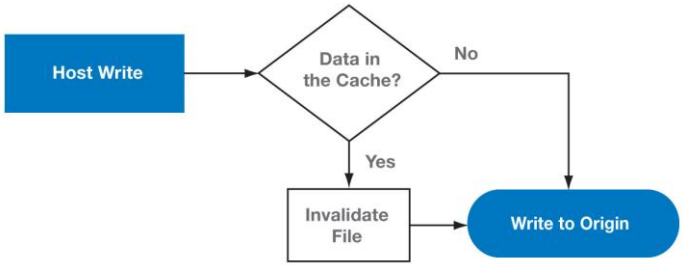

Figure 5) Write from a host.

6 PROTOCOL COMMUNICATIONS

The communications between a FlexCache cache volume and the origin volume is a proprietary NetApp protocol, denoted in Figure 6 as NRV. The protocol is transported over a TCP/IP connection using port 2050.

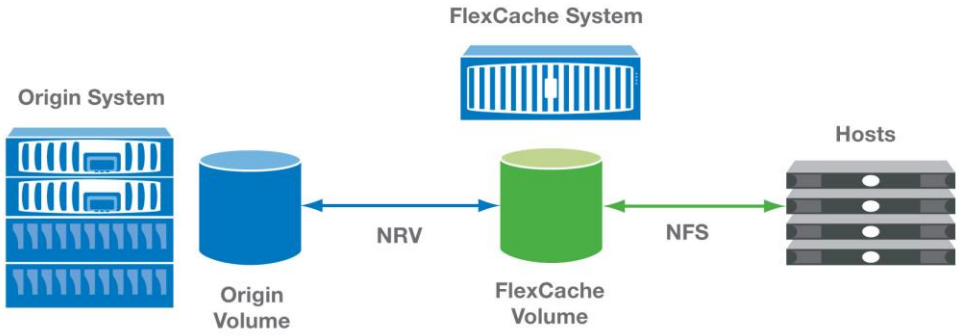

Figure 6) Network communications.

6.1 LOSING THE CONNECTION WITH DATA ONTAP 7.3.0 OR EARLIER

If the version of Data ONTAP on the FlexCache system is 7.3.0 or earlier, the NRV connection between the FlexCache cache volume and the origin volume is critical to the operation of FlexCache. If the connection is unavailable (link failure, origin not available), FlexCache is unable to determine the freshness of data and cannot guarantee its cache consistency policies (including delegations). Therefore, FlexCache is unable to serve data to a host when the NRV connection is unavailable.

Once the connection is restored, the FlexCache system automatically reconnects and begins serving host requests again. However, for previously cached data, the FlexCache system must validate cache consistency before serving data from the cache. With regard to the attribute cache time-out, if it has not expired, the data is still considered valid. Delegations are handled in an all-or-nothing manner. A quick status check with the origin system acknowledges that either (a) nothing has changed with regard to delegations, resulting in all delegations remaining valid; or (b) at least one delegation has changed, resulting in all delegations being revoked on the cache volume.

On a side note, if the FlexCache system is offline or rebooted, the cache volumes lose all freshness context for the data in their cache. All delegations are canceled. This results in the FlexCache system verifying attributes with the origin system for every data request that is in the cache before returning to a steady-state cache system.

6.2 MANAGING THE CONNECTION WITH DATA ONTAP 7.3.1 OR LATER

If the version of Data ONTAP on the FlexCache system is 7.3.1 or later (not including Data ONTAP 10.0.x), then there are options for managing how the NRV connection works when the connection to the origin volume is lost or disconnected. A feature called “disconnected mode” allows the user to enable the FlexCache volume to serve data from the cache when the origin volume is unavailable.

This mode is useful in distributed data environments with unreliable or intermittent WAN connections, where working data sets are not modified often, and remote sites still require access to the data. Disconnected mode allows remote users access to the cached data even if the link with the origin volume is unavailable. For example, if a FlexCache volume caches a `/tools` directory residing at the main data center, `/tools` could remain available to users even if the connection is down. In this example, `/tools` probably does not change often, and if it does, it is not necessary to update the remote site immediately.

When enabling disconnected mode, be aware that data consistency cannot be guaranteed. If the FlexCache volume cannot communicate with the origin volume, data served from the cache may be stale. If your application cannot tolerate stale data, disconnected mode should not be used.

If you are willing to serve potentially stale data for only a specified amount of time, you can set an expiration time for disconnected mode with the `acdisconnected` option. When this option expires, disconnected mode stops serving data until the connection is restored.

Disconnected mode has two modes of operation, based on how your NFS applications handle operations that cannot be completed. FlexCache volumes configured for disconnected mode can respond to read commands only for data that exists in the cache. For other operations, such as changing or writing a file, or modifying the NFS lock on a file, the FlexCache volume responds based on the disconnected mode option setting.

- If disconnected mode is set to soft, then errors are returned to the NFS client:
  - If a requested data block is not in the cache, an error is returned (EIO).
  - On a request to modify a file, an error is returned (EROFS).
  - NLM lock requests are rejected (DENIED_NOLOCKS).
- If disconnected mode is set to hard, then the FlexCache volume drops the request, meaning that the NFS client eventually enters its timeout processing function.
- When disconnected mode is set to off (the default), losing the NRV connection functions as described in section 6.1, “Losing the Connection”.

7 SPACE MANAGEMENT

Space management in FlexCache is designed so that space within an aggregate is efficiently and optimally allocated across multiple FlexCache volumes and FlexVol® volumes. Because of the FlexCache design, very little administrative intervention is required once FlexCache volumes are created within an aggregate.

7.1 SHARING SPACE

An aggregate can contain one or more FlexCache volumes as well as one or more FlexVol volumes. When an aggregate contains multiple types of volumes, it is important to honor *space guarantees* for each volume. This is handled differently for FlexCache volumes than it is for FlexVol volumes. Each FlexCache volume reserves only a small amount of space (by default, 100MB), specified by the `flexcache_min_reserved` volume option. This space guarantee must be honored; if there is insufficient space in the aggregate, the FlexCache volume cannot be brought online. Space beyond `flexcache_min_reserved` is allocated as needed by FlexCache. This allows “hot” volumes to use more space than volumes that are not accessed as frequently.

7.2 RUNNING OUT OF SPACE

During the normal course of operation, as the aggregate fills up with data (from cache volumes as well as from regular FlexVol volumes),, objects in the cache that are not frequently used are ejected in favor of objects that are accessed most often. When necessary, FlexCache selects a cache volume from which to eject data and reclaims space needed for new data. This algorithm favors volumes that are more frequently accessed, and results in least-used volumes occupying less space than frequently used volumes. A FlexCache volume does not eject data within the `flexcache_min_reserved` space.

8 MULTIPLE WRITERS

The FlexCache communications protocol (called NRV, as described in section 6, “Protocol Communications”) is very similar to the NFS protocol. While additional features are built into the NRV protocol for FlexCache communications, with regard to multiple hosts writing to the same file at the same time, FlexCache has the same cache coherency semantics as NFS. This means that if you have a sequence of events, for example, in which two hosts read the same data, then the first host writes some data to the file, then the second host writes new data to the same file, the result is that the file containing that data loses any changes that the first host wrote. This is a property of NFS that must be understood when creating FlexCache for origin volume mappings. If you create an environment in which multiple hosts access an origin volume from multiple FlexCache mappings, you must be cognizant of how your hosts can write to that data.

NFS and NRV provide a mechanism for applications to protect against this “overwriting” situation. If applications are developed in such a way that they use the NFS “file locking” mechanism, the application itself can prevent overwriting from occurring because it limits one host to writing to any one file at a time. FlexCache supports file locking and uses a distributed Network Lock Manager (NLM), just like NFS, to keep multiple processes from simultaneously modifying a file, as well as to coordinate the modification of a shared file between cooperating processes. One other facet of file locking is the use of a Network Status Monitor (NSM) to resolve and reestablish locks if systems and/or hosts go down or are restarted. NSM feedback is supported in a FlexCache environment and is used to resolve the state of the network when appropriate. However, for NFS mounts and host connections in which file locking is not implemented, such as a mount to a shared user directory, users must be careful when modifying files at the same time.

9 CONCLUSION

The FlexCache caching architecture is designed to provide a flexible, reliable, and scalable storage caching system for NetApp storage systems using the NFS protocol. Understanding the techniques and processes outlined in this technical report enables you to enhance and fine-tune your environment for the optimal and most efficient environment for your specific needs.