Technical Report

NFSv3 Enhancements in Data ONTAP 7.2.1

Bikash R.Choudhury, NetApp
July 2011 | TR-3550

ABSTRACT

NetApp has always made conscious efforts to keep improving the NFS stack in Data ONTAP®. Similar efforts were put into Data ONTAP 7.2.1 to enhance parts of NFS implementation in the code to make it more resilient to errors and provide more debugging tools to identify the issues. Changes to the NFS code in Data ONTAP 7.2.1 were aimed at providing better handling of client requests and security than the rest of the NFS vendors provide. There were always challenges whenever an increasing number of clients in the customer base started to stress the architecture of the code. Therefore, the NFS code was redesigned in Data ONTAP 7.2.1 to introduce better access/matching logic, rule sharing, thread changes, new access cache, and enhanced Network Status Monitor (NSM) capabilities in order to improve security as systems scale up with a large number of NFS clients.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 CHALLENGES IN RELEASES OF DATA ONTAP EARLIER THAN 7.2.1

As the number of NFS clients deployed in data centers worldwide keeps growing, improvement in the NFS stack in Data ONTAP becomes more pertinent. The following sections highlight some of the challenges our customers have encountered in their NFS infrastructure and then describe the new 7.2.1 enhancements in the NFS stack that were created to address these issues and make the code more robust.

1.1 GREATER NAME SERVICE DEPENDENCE

Data ONTAP releases prior to 7.2.1 make much greater use of access checking than did earlier releases. In particular, the need to do access checking for all Network File System (NFS) operations, instead of just at mount time, leads to a dramatic increase in demand. This burden is particularly heavy when the controller reboots or does a takeover. This situation is aggravated when there are scalability issues in environments with more clients and export rules in the exports file than expected. In Data ONTAP releases prior to 7.2.1, clients demand access checking only at mount time and have no need to remount just because a controller reboots. After a reboot in in releases prior to Data ONTAP 7.2.1, the controller's access cache is empty, and the first NFS operation from each client must complete an access check before proceeding. Because access checking generally uses slow name services, the hit to throughput can be severe, depending on the amount of host resolution and access cache checking that happens during regular NFS operations.

In Data ONTAP 7.2.1, this issue is addressed by persistently storing access cache information to disk, so that determinations do not have to be redone after a reboot, takeover, or giveback access. Also, access determinations are done more efficiently and are scalable due to a variety of improvements in export handling.

1.2 REFUSAL OF ACCESS FROM NAME SERVICE FAILURE

Name service failure causes an avoidable refusal of access. Some export rules are disjunctive. For example, `ro,rw=group1` permits read-only (RO) access for any client or read-write (RW) access if the client is in group1. If host1 is not in group1 and name service is working, the rule evaluates to `ro=true` or `rw=false`, and that simplifies to `ro=true`. If name service fails, we have `ro=true` or `rw=unknown`. In releases prior to Data ONTAP 7.2.1, any unknown value causes the whole rule to be unresolvable. Also, identical export rules are evaluated again and again, which leads to bigger problems with more virtual volumes (vvols).

In Data ONTAP 7.2.1, read-only access can be granted in such situations even when the question of write access is not resolvable. Also the “rule-sharing” feature allows each rule to be evaluated only once for a given address throughout the duration of the session in which the results apply to each export that uses the same rule.

1.3 NO PROVISIONAL ACCESS ON MOUNT

No provisional access on mount should be allowed. Provisional access is granted when a name service outage prevents the controller from determining whether a given client has access to an exported path.

In order to address some of the export-handling issues in releases prior to Data ONTAP 7.2.1, the NFS code contained a provisional access feature. An option could be turned on that caused evaluation of export rules to proceed based only on IP-related elements of the export rule, ignoring all other elements. The problem with this approach is that evaluating a rule partially may grant access incorrectly and in unexpected ways.

In Data ONTAP 7.2.1, such provisional access is disabled. In cases in which provisional access was assured of providing a correct answer, the new export evaluation logic provides that correct answer by

simply avoiding the evaluation of parts of the rule that can be determined not to affect the final result. The `nfs.export.allow_provisional_access` option is removed or has no effect.

1.4 INEFFICIENT EVALUATION OF EXPORT RULES

Evaluation of export rules in releases prior to Data ONTAP 7.2.1 is inefficient. The evaluation does more work and waits for additional data that is not needed to reliably determine whether access is granted. This is fixed by the new access/matching logic introduced in Data ONTAP 7.2.1.

1.5 TOO MANY THREADING CHOKEPOINTS

There are too many threading chokepoints while processing mount and NFS requests from the client. Access determination is single threaded. The access cache blocks further NFS requests until the mount assist thread sends or times out on a name resolution. Therefore, the requests tie up threads while waiting for access determination.

Similarly, the mount handling is single threaded and blocks further mount requests until the name resolution is successful.

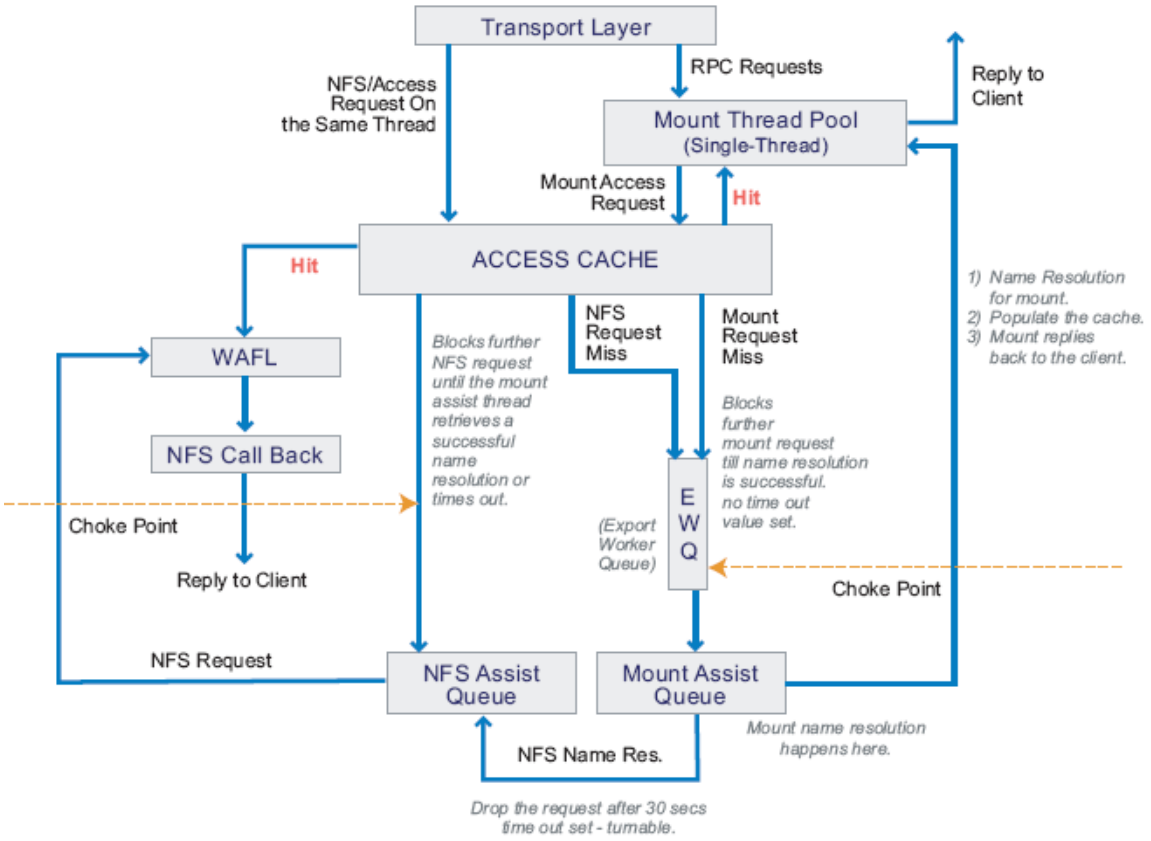
The export/mount threading redesign in Data ONTAP 7.2.1 fixes these issues, allowing multiple mount requests to be processed in parallel and multiple export determinations to be made in parallel.

1.6 ACCESS CACHING ISSUES

In the existing architecture, the access cache is lost after a controller failover/giveback or a controller reboot. IP/subnet exports are not optimized in the access cache. The access cache hit path is not efficient because of hash table implementation. The Read and Write determination is done together, resulting in bugs and performance issues even though the file system has a Read Only rule in the `/etc/exports` file.

The new access cache design and implementation in Data ONTAP 7.2.1, which includes the persistent storage of access results, help improve access cache behavior. Figure 1 shows the earlier architecture of Data ONTAP, before 7.2.1.

Figure 1) Architecture of Data ONTAP earlier than 7.2.1.



2 EXPORT RULES

The export rules provide a means to define which hosts are allowed to access the various NFS server exported trees and to limit the type of access that each host is allowed, based on an export specification for that export.

For a given export, one or more export rules may be set. Each such export rule applies to all accesses made to the given export with a specified set of security types. The same export rule may be associated with all allowed security types, or different export rules may be associated with different security types. The export rule defines the hosts that are allowed various forms of access to the given export. Depending on the export rule, a given IP address may be denied access, or it may be allowed read-only access or read-write access. Also, specific IP addresses may be allowed or not allowed root access (that is, access by UID 0). If root access is not allowed, the export rule specifies under what UID, if any, such access can be performed.

NFS and mount protocol requests generally require an access determination to be made, to be certain that the host making the request has the appropriate level of access (for example, read or write) for the request being made. This access determination is in addition to normal file-level access determination, which determines whether the user making a request has the appropriate level of access to the controller directory being acted upon. Each request contains the requesting host's IP address and a specification of the export being acted upon (in the form of a path for mount requests and a file handle for NFS requests). The controller must determine that the client host has permission for the appropriate type of access before the request is granted. The determination is made by consulting the export rules in effect at the time of the request. Each rule specifies which clients can access a given export, specified in terms of the

root path of that file tree, and what types of access they are permitted. The export rules allow clients to be specified indirectly by subnet, domain name system (DNS) name and subdomain, and netgroup. Because the request has only the IP address, the controller must resolve the names given in export rules before access can be granted or denied, and that usually involves name service requests.

Name service can be slow or temporarily unavailable, so performance and responsiveness can be improved by avoiding name service requests unless necessary, or by doing them in advance of the request. The Data ONTAP 7.2.1 NFS code includes a number of improvements that address these goals.

The exports cache is used during NFS and mount operations to determine whether a client's access to an exported portion of the file system has recently been determined, and, if so, what that result is. Clients are identified exclusively by their network addresses.

The new access cache implementation uses a radix tree to store the cached data. The radix tree is very efficient at storing hierarchical information, such as IP addresses within a domain. In particular, the radix tree is very efficient at representing subnets. Other aggregations, such as netgroups and domain names, can also result in heavy concentrations of similar IP addresses that have the same access results. The combining operation can be implemented in the radix tree to consolidate those nodes, reducing both access time and memory consumption and increasing the likelihood of hitting the nodes in CPU cache.

2.1 RULE SHARING

In the earlier Data ONTAP releases, the exports checked hosts for write access multiple times, with the number of checks increasing as the number of clients and exports increased. However, in Data ONTAP 7.2.1 the rules and the access cache are shared. For example:

```
/vol/one ro,rw=@fred
```

```
/vol/two ro,rw=@fred
```

Access determination made for one volume applies to the other as well. This reduces space and CPU requirements, with less need for name services access.

Rules must be identical to be shared. For example:

```
rw=@foo:@bar is not identical to rw=@bar:@foo
```

Rule sharing also extends to security type.

Suppose that you have this:

- `/vol/one sec=sys,rw=@fred,sec=krb5,ro,rw=@fred:joe`
- `/vol/two sec=krb5,rw=@fred`
- `/vol/three sec=sys,ro,rw=@fred:joe,sec=krb5,rw=@george`

Then you have sharing for this:

- `AUTH_SYS(/vol/one) and KRB5(/vol/two)`
- `KRB5(/vol/one) and AUTH_SYS(/vol/three)`

And you have no sharing for `KRB5(/vol/three)`.

2.2 RULE EVALUATION

In the releases of Data ONTAP earlier than 6.5, rule evaluations were done by a multipass process. This algorithm evaluates the IP addresses and then the host names, followed by the netgroups, which are the most expensive.

If you have the following rule:

```
1.2.3.4:fred:@foo:5.6.7.0/24:@bar:george
```

Then you never need the host name and you never need to evaluate a netgroup if you are looking for IP address 5.6.7.8. If you have 5.6.7.8, then the list is not all IP addresses. The algorithm does a reverse map from 5.6.7.8 to name (suppose the name is “tom”). IP address 5.6.7.8 is not 1.2.3.4. The name doesn’t match “fred.” The algorithm then checks for membership in netgroup “foo” and finally checks whether 5.6.7.8 is in the subnet 5.6.7.

After Data ONTAP 6.5, this multipath algorithm was replaced with a single-pass algorithm, because, with the new features, there was no obvious way to evaluate the rule out of order. Such new features as negated entries and interacting `ro=` and `rw=` lists resulted in a simple (but inefficient) single-pass algorithm, which often made unnecessary requests of name services and was unable to make access determinations, even though the information to make those determinations should have been available.

In Data ONTAP 7.2.1, rule evaluation uses a single pass for read and write and evaluates both `ro=` and `rw=` lists if they are present. For each list, the algorithm can do a simple evaluation if the list is only IP based; otherwise, a reverse-map IP to name happens, and it goes through the list in order. This avoids unneeded work and returns an answer whenever needed, and it also avoids waiting for data that can’t affect the result. It does not do reverse-mapping of IP when forward-mapping of name is more efficient. It uses a multipath algorithm but provides the same semantics as in releases prior to Data ONTAP 7.2.1.

The new multipass match does only the simple things in early passes. It keeps track of possibility sets; hosts and IP addresses that are skipped are not ignored. Finally, when the match is found, the algorithm exists.

If there are hostnames, the possibility sets are combined for RO/RW interaction, and we get the IP address. If they aren’t combined, we note that fact and assume that they are netgroups. Such preresolutions are periodically refreshed. However, if name services are down, we may not have the information. Also, access determinations are frequently delayed because any period of name service availability results in the appropriate preresolution being done and saved for later use.

Following are some of the examples of the new export evaluation mechanism that is implemented in Data ONTAP 7.2.1.

Example 1:

If the client has an IP address 5.6.7.8 and the rule is

```
ro,rw=1.2.3.4:fred:@foo:5.6.7.0/24:@bar:George
```

it goes through the first pass:

Start with {no-match}.

1.2.3.4 doesn’t match, so it is still {no-match}.

For hostname, IP is fetched in the background.

- If IP is fetched, we still have {no-match}.
- If name services are down, fred could be hostname or netgroup; we don’t know.
- So we have {no-match,group,single}.
- Skip netgroup, so add {no-match,group} to the set.
- So we have {no-match,single,group}.
- Now we have the subnet, and it does match.
  - No-match is not possible.
  - Group match would happen if we get this far.
  - But we might have had a single match on fred.

  - Result is {single, group}.
- RW access is granted in either case, so we are finished.

Example 2:

If the client has the IP address 5.6.7.8, and the rule is

`ro=5.6.7.0/24,rw=fred,5.6.7.0/28`

In the first pass, RO list results in {group}.

RW list is normally {group}.

- Assuming that fred's IP is fetched (not 5.6.7.8)

But if name services are down:

- If down since export was done, we have {single,group}.

Can decide only read access; write access determination has to wait.

Example 3:

If the client has the IP address 5.6.7.8, and the rule is

`ro=5.6.7.0/24,rw=-fred,5.6.7.0/28`

In the first pass, RO list results in {group}.

RW list is normally {group}.

- Assuming that fred's IP is fetched (not 5.6.7.8)

But if name services may be down:

- If down since export was done, we have {neg-single, neg-group,group}.

Are not able to decide any access.

2.3 BEST-PRACTICE RECOMMENDATIONS

Best Practice

- **Take advantage of rule sharing.** Whenever possible, make the rules for multiple exports exactly the same rather than almost the same. Note that a rule encompasses the `ro`, `rw`, and `root` list, as well as things like the `anon=` list. If one volume has `root=a` and another has `root=b` and they are otherwise the same, consider specifying both as `root=a:b`, if you can do that and remain consistent with your security requirements.
- **Always set the strict netgroup option and specify netgroups using the leading @ sign.** The negative effect of not doing so in Data ONTAP 7.2.1 is smaller than in previous releases, but it has not been eliminated. The negative effect is likely to be limited to cases in which there are name-service problems, so you might not see the problems until you are in a bad situation, and not having set strict netgroups makes a bad situation worse.
- **Form the rule specifications as lightweight as you can, consistent with your needs.** The element types in an export specification, from easiest for the system to deal with to the most difficult, are these:
  - IP addresses or subnets (require no name services to resolve)
  - Hostnames (require DNS mapping only periodically to convert hostname to IP)
  - Subdomains (require reverse DNS mapping for each new client)
  - Netgroups (require reverse DNS mapping and netgroup membership checking for each new client)

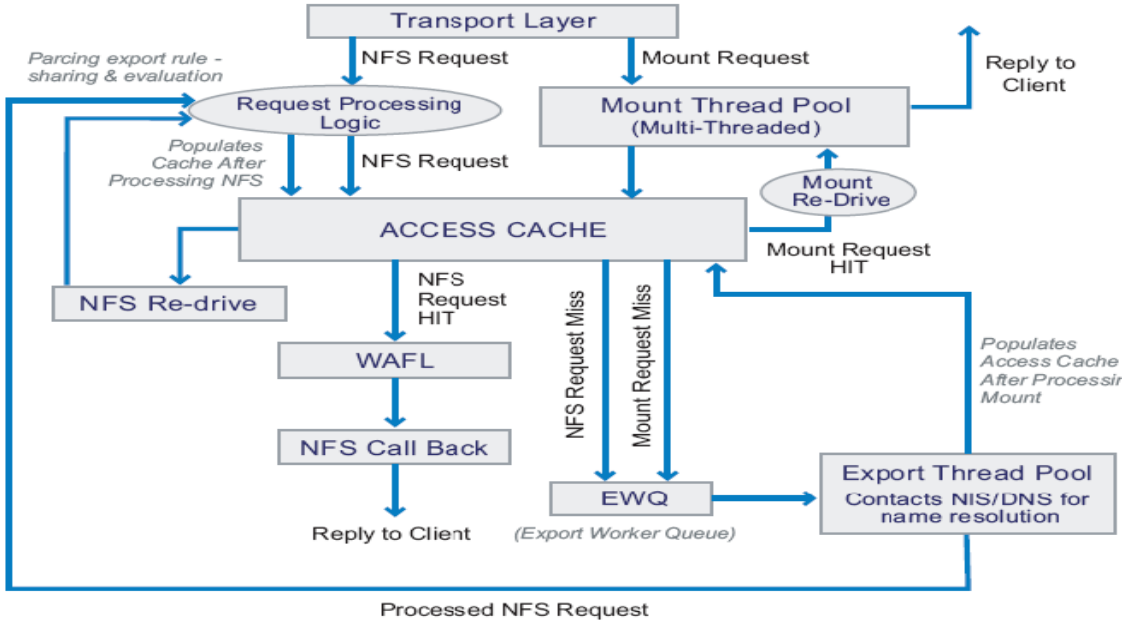
Note: In these best practices, specifying multiple subdomains in a list does not take any more effort than specifying one; but when netgroups are used, each additional netgroup in a list takes an additional increment of work.

- **Even when netgroups are necessary, consider whether certain classes of clients can be resolved using IP addresses only.** If possible, specify a subnet for these to reduce netgroup membership tests.
- **Use negated entries sparingly because they can complicate the rule evaluation process.** Negated netgroup specifications have the most severe effect, because they typically mean that netgroup membership checks must be done for all access determinations.
- **Evaluating both an `ro=` list and an `rw=` list for the same export adds to evaluation complexity, particularly when one or both of the lists contains netgroups.** Consider whether a uniform level of access is possible. Note that wildcards (RO or RW without the equal sign) do not have the same effect.
- **Consider application patterns when setting the export harvest time option.** If there is a set of batch jobs in which a large set of clients access sets of exports every 45 minutes, then the harvest time should probably be set so that entries do not leave the access cache between iterations of these jobs.

3 ACCESS CACHE

Figure 2 shows how Data ONTAP is structured in release 7.2.1.

Figure 2) Architecture of Data ONTAP in 7.2.1.



The access cache is used during NFS operations to determine whether a client's access to an exported portion of the file system has recently been determined, and, if so, what that result is. It is also used during mount operations. Clients are identified exclusively by their network addresses, except when there are hostnames in the export rules that need a name resolution. The cache is a reservoir of access results. Individual results are accessed by a number of keys. Some keys are used to divide the systemwide access resolution cache into rule-specific caches. The remaining keys are used to locate individual cached access results within a rule-specific cache (rule cache).

This is the complete set of keys for lookup:

- vFiler® unit ID
- Client address
- Export point

The access results cached for each combination of keys include whether each of the following is positive, negative, in progress, delayed, or uninitialized:

- Read access
- Write access
- Root access

An attribute is scheduled for refresh when there is a cache hit or if the attribute was inserted in cache for more than the timeout value set. The cache attributes that get refreshed are categorized as POSITIVE, NEGATIVE, or DELAYED.

The following options control the refresh process:

- `nfs.export.neg.timeout 3600`
- `nfs.export.pos.timeout 36000`

Stale access information is returned until the refresh happens.

DELAYED information expires after 15 seconds (can't be changed).

A node is scheduled for harvest when it has not been accessed for longer than the timeout value specified (in seconds). Harvest removes the access node from cache.

The following option controls harvest:

- `nfs.export.harvest.timeout  1800`

IN-PROGRESS nodes are not harvested.

Releases of Data ONTAP earlier than 7.2.1 have the following features:

- There is one access cache per export rule. (This is still true in Data ONTAP 7.2.1.)
- Access permissions are cumulative; when all requested attributes are found, cache lookup is finished. RW is always done together, even though the export rule states RO for the file system.
- The access cache hit path is stored in 256 hash chains. In very large client farms, the lists may grow unacceptably large.

Multiple access caches exist for different security types, even for the same export. For example, for `/vol/volA –sec=sys; krb; krb5i`, there are separate access caches for `sys`, `krb,` and `krb5i` for the same rule.

- All of the access cache information is stored by IP address.

All name resolutions happen on the mount assist thread.

- Access cache information is not persistent during controller reboots and cluster takeovers and giveback operations.

Data ONTAP 7.2.1 contains several improvements to the access cache. The new cache has the following characteristics:

- Fast resolution in synchronous code paths of whether an access result is known, and, if so, what the answer is
- Separate determination of whether a client has read and/or write access, and whether or not root ID translation is in effect for that client
- Rapid restart after crash and reboot or failover
- Caching of positive, negative, and mixed positive and negative results
- Fast insert into the cache
- Bounded search time
- Continuous cache maintenance in the background
- Optimized handling of subnets
- Storage in a radix tree
- Storage by individual IP address as well as subnets
- Representation of 1 bit of IP address on each level of the radix tree
- Facilitated queuing of a request in case of cache miss
- Use of an export thread pool for processing cache misses
- Persistence (tunable)
- No IPv6 support
- Potential difficulty with compliance because persistent copy does not make a provision for IPv6

The access resolution cache is designed to be stored persistently on disk. The purpose of storing the cache contents persistently is to improve the controller's response time during the period after a restart.

The radix tree is very efficient at representing subnets in particular. Other aggregations, such as netgroups and domain names, can also result in heavy concentrations of similar IP addresses that have the same access results. The combining operation can be implemented in the radix tree to consolidate

those nodes, reducing both access time and memory consumption and increasing the likelihood of hitting the nodes in CPU cache.

3.1 NFS REDRIVE

NFS requests that result in access cache misses are linked to their corresponding cache population request until the IP lookup for these requests is complete. When the lookup is complete, the requests can be processed and are “redriven” through a different queue called the redrive queue.

3.2 PERSISTENCE OF ACCESS CACHE

In the new Data ONTAP 7.2.1 code, the access cache contents are persistent across controller reboots and cluster takeover and giveback. The cache contents are dumped to disk every 15 minutes. One file per export rule is saved, named by ruleid. The ruleids are unique while the controller is up. The files are saved as:

- `/etc/exports_arc/cache`
- `/etc/exports_arc/restore`

Upon reboot or failover, cached information is restored and inserted back into the cache entry, greatly expediting the cluster takeover/giveback process. The persistence file contains:

- Rule definition
- Access cache nodes
  - IP address
  - Positive and negative access information

There is an option to turn persistence off: `Options nfs.acache.persistence.enabled off`

3.3 NEW CODE SEQUENCE

In the new Data ONTAP 7.2.1 code, the sequence of the request path is as follows:

NFS/Mount request comes in from IP [A].

- Request requires read/write access.
- Lookup in the cache for RW for IP [A].

  Cache hit:
  - RW access information returned to caller.

  Cache miss:

    A node is created for IP [A] if not already present in the Request Processing Logic area.

    Request is queued on that node waiting for RW attributes to be inserted.

    State for RW attributes is set to IN PROGRESS.
  - Access matching layer determines RW access for IP [A].
  - Request is sent on the name resolution thread pool to resolve the name. This is done in the Export Thread Pool.
  - After the name resolution, checks whether the access provided is based on the rule.
  - Request processing layer inserts information in the cache.

    State for RW attributes is set to Inserted.
  - Queued request is returned to caller.
- Request is processed and reply is sent to IP [A].

In case of a cache hit, the sequence of events is as follows:

Cache contains RW information for subnet 192.100.10.0/24 for export [A].

Client 192.100.10.1 tries to read a file under export [A].

The client is in the subnet, so RW information for the subnet node is returned.

- Node for 192.100.10.1 is not created.

In case the name server is down:

Access cannot be determined because of name service outage.

- DELAYED is inserted in the cache.
  - EJUKEBOX is returned to the caller (v3 and v4).
  - Request is dropped for v2.

4 MOUNTD AND THREADING

There is a difference between MOUNT and mountd. MOUNT is a client-side command, and moutd is the daemon responsible for mounting the NFS share that is exported from NetApp® storage. The MOUNT is next in line after portmap for interaction with the client. The mount checks the client requests for the permissions assigned on the share before the NFS layer grants a filehandle and completes the mount successfully. The RPC program number for mount is 100005.

In the releases of Data ONTAP earlier than 7.2.1, mountd is a single-threaded process that services each mount RPC request one at a time. When an NFS request comes in that requires a name lookup, it is immediately added to the assist queue, along with the name lookup on the mount assist thread that happens at the same time. Therefore, it is possible for a single request to be responsible for blocking two threads at once. There are three reasons to do name lookups. In order of priority, they are mount requests, NFS requests, and cache refreshing. At this time, there is no way to favor the higher-priority requests without allowing mount storms to completely freeze out NFS requests.

4.1 CURRENT LIMITATIONS OF MOUNTD IN DATA ONTAP

MULTIPLE THREADS BLOCKING

This design can lead to two threads being blocked at the same time waiting for the same event. For mount requests, the requesting mount thread blocks while waiting for the mount assist thread, which itself can block while waiting for a name resolution request completion.

The NFS assist thread can also be blocked waiting for the mount assist thread to complete a network request.

REQUESTS THAT COULD RUN ARE BLOCKED BY EARLIER REQUESTS

While these threads are waiting, requests that might otherwise be completed can be stuck behind them. In particular, incoming mount requests for which the required access information is present in the cache cannot be processed while the mount processing thread is blocked waiting for the mount assist thread.

This problem is less severe for NFS requests because those whose access rights can be determined by current cache contents can proceed on the Ethernet driver thread. However, there can be access cache misses that could be satisfied immediately from cached information available in the network layer; these requests end up waiting in the queue for the assist threads while they in turn wait for mount assist threads.

SERIALIZATION

There is one mount assist thread per vFiler unit, so all of the name resolution work is done on a single thread. Therefore, the resolution for one request cannot begin until the preceding request is completed.

This means that the latency incurred for a flurry of requests is additive, because the last request in the flurry incurs the latency of its predecessors in addition to its own.

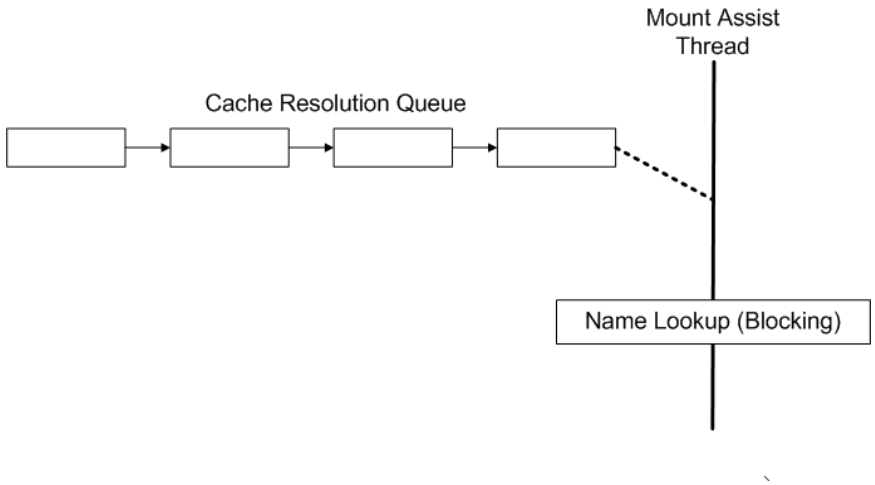
PRIORITY

Mount, NFS, and cache refreshing all have the same priority because they are not distinguished from each other by the mount assist processing. In particular, refresh should not interfere with mount or NFS requests.

QUEUING ANOMALY

Part of the preceding serialization statement is untrue, because of what is presumed to be a programming error in the current implementation: Queue addition is at the front. Therefore, the latency of a request, rather than being the sum of its own plus its predecessors' latency, could be the sum of its own and its successors' latency. Furthermore, a continual stream of requests can block an earlier request indefinitely. Figure 3 shows a diagram of the queuing process.

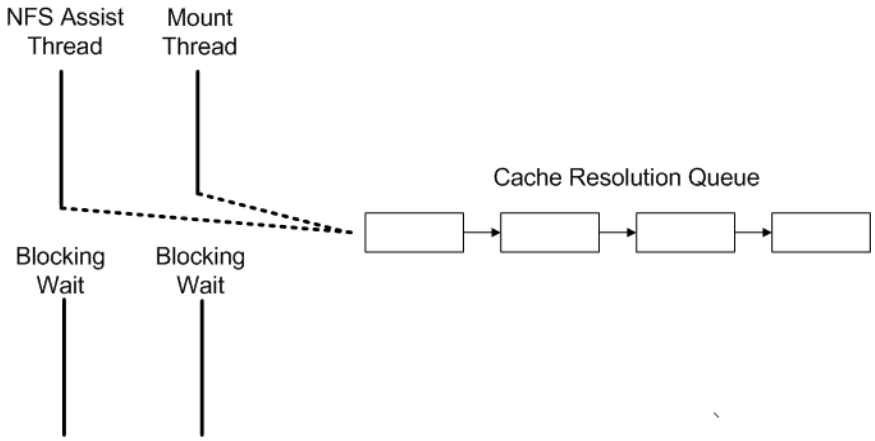
Figure 3) Queuing process.



A request that is waiting for a name service lookup should not block requests that don't need one. This is easier for NFS requests than for mount requests because NFS requests already have a scheme to move blocking requests elsewhere. These lookups involve blocking network requests. These queries cannot be made in line (serially) because NFS requests are processed on the Ethernet driver thread (through upcalls from the network), and blocking is not permitted on this thread. Also, the blocking would interfere with other requests that would otherwise be serviced.

This blocking request is only rarely needed, because the name lookup results are cached, and most requests proceed normally on the Ethernet driver thread after access is granted based on the cache lookup. Only in the case of a cache miss is the name lookup needed. When the need to make the name server query becomes apparent, the work is shifted to other threads. The name request is put on the Export Worker Queue (EWQ), serviced by the mount assist thread (one per vFiler unit). The NFS request is put on the assist queue, serviced by the NFS assist threads (two of these, total). Figure 4 shows a diagram of these threads.

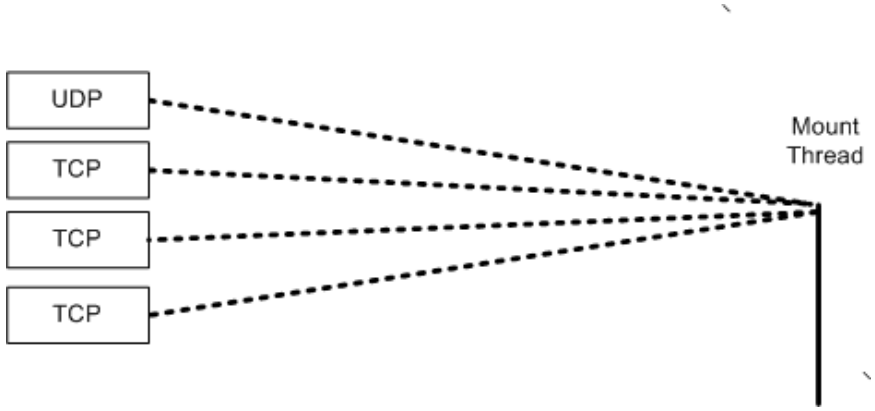
Figure 4) Alternate threads for name server query.



When it receives the queued NFS request, the NFS assist thread simply runs the same code that was run before on the Ethernet driver thread. The idea is that the name request has been processed and the results put into the cache by the time the NFS assist thread processes the NFS request. It then finds the proper cache entry and continues processing the request.

Of course, this is not guaranteed to work because the mount assist thread may not yet have processed the EWQ entry, or it may not yet have received a reply. Therefore, the NFS assist thread does the name query itself if it experiences a cache miss and then proceeds with the request. (The NFS assist thread is allowed to block.) Figure 5 shows a diagram of the mount thread.

Figure 5) Mount thread.



However, this behavior is significantly changed in Data ONTAP 7.2.1 to better handle the name lookup process by the mount assist queue using a more enhanced thread pool package. Mount request processing is no longer single threaded. Mount and NFS requests that can be processed are no longer potentially blocked by requests that cannot be processed.

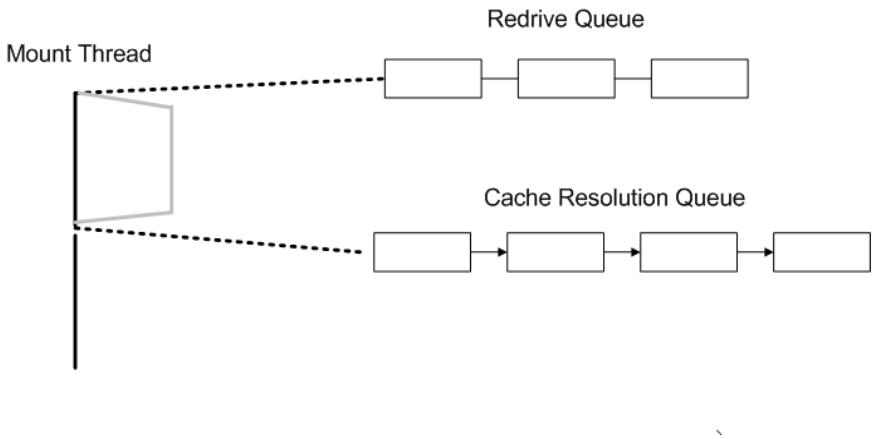
Mount Redrive

The mount redrive component facilitates parallel mount request processing. Mount requests that result in access cache misses are linked to their corresponding cache population request until the IP lookup for these requests is complete. When the lookup is complete, the requests can be processed and are redriven through the mount redrive queue. A pool of threads processes requests off this redrive queue.

The important point is that the mount process no longer blocks to wait for the cache to complete the reverse name lookup resolution. Instead, the thread that is processing the request is freed up to work on

other mount requests, and the original request is then requeued to the mount thread pool when it is ready to try again. Figure 6 shows a diagram of the mount redrive process.

Figure 6) The mount redrive process.



The new mount design introduced a new exports thread pool. If we had a lookup function, we could simply call it, put the current resolution request aside into the exports thread pool, and continue to process the next one, and resume the original request when the asynchronous result arrives. This enables us to have maximum parallelism for lookups using this function asynchronously. Each cache resolution request runs on its own thread and blocks in `gethostbyaddr_r`. We get some parallelism simply by having more than one thread dedicated to this process.

This changes the mount request processing to eliminate the blocking that currently happens while the I/O request is waiting for cache population to occur. The result is not a true multithreaded mount processing, but it does have the benefits of multithreading when dealing with delays caused by access determinations.

Note: Clients that are still mounting the file systems from the controller using UDP cannot benefit from the new multithreaded mount processing; requests from those clients still use single-threaded operations. Clients mounting with TCP benefit greatly from this enhancement.

Therefore, the new mount thread pool handles multiple mount threads where the requests wait in queue for a free thread. Unlike releases of Data ONTAP earlier than 7.2.1, where the mount assist thread is stalled for 500ms for every name lookup, this condition has been fixed in the new code.

5 NLM AND NSM

Network Lock Manager (NLM) is an ancillary protocol of NFS that supports file and record locking for files accessed over NFS.

Network Status Monitor (NSM) is an ancillary protocol of NFS that defines the protocol required to notify the NLM clients about lock reclamation in the event of an NFS client/server reboot.

NFS was meant to be a stateless protocol. The decision of the inventors (and later the standards body) was to leave the inherently stateful locking semantics out of the mainline NFS protocol. Therefore, the locking of NFS files was implemented with the help of two more protocols, NLM and NSM. NLM defines a protocol for locking and unlocking NFS files. NSM defines a protocol whereby interested hosts can be notified of a change in state of a particular host. When NSM is applied to NFS, it helps in timely notification of client or server loss of lock state. The clients are then allowed by the controller's NLM to gracefully reclaim the locks.

To summarize, NLM coupled with NSM allows a seamless recovery of lock state in case of catastrophic events like controller crash and reboot. NLM typically goes into a grace period recovery immediately after a reboot to allow only the lock reclaim requests. There are some edge conditions under which the lock reclamation might fail; for example, if the lock reclaim request couldn't make it to the server in the given grace period.

Currently, the Network Status Monitor on the controller does the following activities:

- NSM makes lock recovery happen.
- NLM does not persist in individual lock state.
- The client identifies itself with a unique ID (typically a FQDN) in the NLM request.
- The unique ID is supplied to NSM for monitoring (that is, adds it to its persistent repository).
- NLM relies on NSM to maintain client lock state.
- NSM maintains a persistent repository of client lock state in `/etc/sm/monitor`.
- If a server crashes and reboots, NSM:
  - Consults the `/etc/sm/monitor database`
  - Copies the active entries in `/etc/sm/monitor` to `/etc/sm/notify` and removes `/etc/sm/monitor`
  - Starts issuing `SM_NOTIFY RPC` requests to clients to reclaim the locks
  - Meanwhile, goes into grace period recovery
  - While in grace period recovery, accepts only reclaim requests (reclaim bit in NLM request).
  - Denies new requests with an `NLM_GRACE_DENIED` error during the grace period recovery

For releases of Data ONTAP earlier than 7.2.1, the issues pertaining to this behavior are described in the following sections.

5.1 NSM STARTED BEFORE NLM

The NSM thread is started before NLM. This leads to a race condition in which the client is notified and attempts to reclaim locks before NLM registers with the portmapper. HPUX11.11 and Linux® 2.4 are observed to attempt the portmap lookup only once, without retrying.

On the client side, there are also issues in handling NSM and NLM:

- The AIX 5.x and Solaris 8/9 clients are the most robust for handling locking in the face of server failures and notification.
- HP-UX 11.11 and, to a lesser extent, 11.23 have problems with response time and robustness.
- Linux 2.4 with stock `nfs-utils` doesn't work. Linux 2.6.13 just doesn't work.
- Data ONTAP can mitigate the problems with HP-UX and Linux.

5.2 SINGLE-THREADED NOTIFICATION

Notifications are conducted sequentially. The current algorithm sends the notification and waits 100ms for a reply, repeating up to five times before proceeding to the next client. Coupled with slow notification response from Linux and HP-UX clients, this results in slow or missed notifications from the controller.

5.3 NAME RESOLUTION TIME

Each client name is stored in the notification database and must be resolved before notification can be sent.

UNBOUNDED NOTIFICATION TIME

Notifications from nonresponsive clients take a long time. When the fifth notification try fails, the thread sleeps for 10 seconds, and the clients that did not respond are retried. Clients that are down or on a partitioned network are not notified. Clients that reboot on the network send a NOTIFY to the controller that appears to be properly handled.

For every host in `/etc/sm/notify`:

- Resolve the client FQDN.
- Check whether the NSM service is listening (portmapper).
  - Next step, if client responds
  - (else) UDP request retried 20 times with 500ms timeout and then continue
- Send the NSM notification.
  - Mark notified, if client responds
  - (else) UDP request retried 20 times with 500ms timeout and then continue

PASSING UNDECORATED SERVER NAME IN NOTIFY

Each NOTIFY to clients includes the undecorated server name. This is perfectly OK, but the Linux and HP-UX clients appear to resolve the name before responding to the NOTIFY. Algorithms that use synchronous response to notifications, such as the current version, suffer from this.

LINEAR ACCESS TO NOTIFICATION DATABASE

The database format for notification is searched linearly. There is no provision for IP address in the database record.

Table 1 summarizes behavior organized by type of operating system.

Table 1) Summary of behavior by OS type.

	AIX	Solaris	HP-UX	Linux 2.4 nfs-utils-101	Linux 2.4 nfs-utils-107	Linux 2.6 nfs-ytuks-107
Client format for server names	FQDN	FQDN	FQDN	Dotted quad	Dotted quad	Dotted quad
Response time to NOTIFY if server in resolv.conf	Short	Short	Long	Long	Long	Long
Reclaim works if server not in resolv.conf.	Yes	Yes	Yes	**No** statd uses unprivileged port	Yes	**No** lockd does not recognize server name
Reclaim works if server not in resolv.conf.	N/A	N/A	N/A	N/A		N/A
Reclaim works if server sends FQDN and server not in resolv.conf				N/A		N/A

	AIX	Solaris	HP-UX	Linux 2.4 nfs-utils-101	Linux 2.4 nfs-utils-107	Linux 2.6 nfs-ytuks-107
Reclaim works if server sends dotted quad	No	No	Yes	N/A	Yes	N/A
Reclaim requires NLM active before NSM	No	No	Yes	N/A	Yes	N/A
Frequency of NLM portmap retry	<10ms	2 sec	None	None	None	None
Behavior reclaiming lost locks	Reclaim denied, application unaware	Reclaim denied, SIGLOST sent to application	Reclaim denied, SIGLOST sent to application	Reclaim denied, application unaware	N/A	N/A

In Data ONTAP 7.2.1, the code design addresses many of these limitations.

- NSM databases store the IP address of the client along with the client FQDN.
  - Less dependent on name service
  - If IP does not work, then switch to FQDN
- NSM notifications send in batches of 1,000 requests.
  - One NSM clock tick (100ms) is exhausted for 1,000 hosts.
    - ☐ Best case: 100,000 hosts = 100ms * 100 = 10s
  - Responses are received out of sync with requests.
- Unlike in releases of Data ONTAP earlier than 7.2.1, where the NLM callbacks were synchronous, partly because name resolution was synchronous, the scenario changes in Data ONTAP 7.2.1. The name resolutions are multithreaded in the sense that multiple NLM callbacks can be processed in parallel. Even if this is still over UDP, it is now asynchronous.

Because UDP is unreliable, an attempt is made to retry each packet up to five times before giving up. The basic timing tick is relatively short, about 100ms. (This and some other numbers given can be made into parameters for tuning purposes.) Each state is retried up to five times before timing out. Thus, a pathological client could require 400ms per state to complete. However, the expected behavior is either rapid completion or no response to the portmapper request, with occasional dropped packets resulting in delays of 100 milliseconds. An unresponsive client would be removed in 500ms. For a 20,000-client configuration, with all clients unreachable or down, it would take 20 sets of timeouts to pass through all the clients, indicating a worst-case time of 10 seconds to completely fail. This approach does a better job of producing progress by guaranteeing timely retries of the UDP transmissions at each stage of the overall notification, and the detailed design uses this mechanism.

- NSM notifications are sent for every controller hostname.
  - Client could have stored any of the hostnames in the persistent database.
- There is a smart XID generation algorithm.

If we are notifying in parallel, we need a discriminator in the return from the client that allows speedy translation to the originating record. We can use the return XID from the client as an index for direct lookup. Normally, a generic XID allocation routine is used because we want to use the XID only as a

unique tag rather than using it to convey information. In this case, we control the XID number to our purposes.

Different vFiler units on the same physical controller can use the same XID, because the XID is scoped by the IP address.

- We create our own XIDs rather than relying on the system defaults in order to use the returned value as the index.
  - Higher 22 bits = NSM clock tick
  - Lower 10 bits = byte index into the database (for marking it monitored/unmonitored)
- Until all entries are notified in `/etc/sm/notify`
  - Notify the first 1,000 entries in /etc/sm/notify
    - ☐ Callback XID has current clock tick and index into database
- Wait for response until the expiration of the current NSM tick (100ms).
- Pick the next 1,000 entries and restart the procedure.
  - Remember, the clock tick also changes.
  - If earlier clock tick replies arrive late, they are discarded.

6 REVISION HISTORY

Table 2) Revision history.

Date	Comments
July 2011	Updated to current template
January 2007	First release

Go further, faster®

www.netapp.com