Resilience
(aka Replica Manager, Next Generation)

Albert L. Rossi

Fermi National Accelerator Laboratory

Current Replica Manager

→ ***dCache Book,*** **Chapter II. 6.**

- https://www.dcache.org/manuals/Book-2.15/config/cf-repman-fhs.shtml

◆ ***replica service*** (**Replica Manager**) controls number of replicas of a file on the pools.

◆ for durability and availability in absence of tertiary file system:

- Uses p2p to guarantee number of replicas is at least the minimum (2 is default).
- If more than maximum (default is 2) replicas exist, some of them will be deleted.

◆ hybrid mode (one resilient pool group per manager instance, multiple non-resilient groups)

Current Replica Manager

- No significant modification since 2007.
  - Maintains a rather heavyweight set of database tables (written before move to Chimera).
- Handles only NEARLINE files that are not allowed to be stored to tape by using lfs=precious settings on a pool (pre-dates RetentionPolicy/ AccessLatency concepts).
  - Intrinsically incorrect
  - lfs=precious is a legacy option
- Limitations/brittleness:
  - Allows for only one resilient pool group per instance.

    To simulate the existence of different resilient groups, one has to run as many Replica Managers as the pool groups one wants to make resilient.
  - Replica range is fixed to min <= n <= max for *all* files in the group.

Re-evaluation

- Need to continue to provide replication at the application level, and make use of it within a larger set of QoS requirements.
- It must be more configurable and flexible than the current manager.

Principal Requirements

1. No database to maintain or synchronize with the namespace.
2. A single service instance must be able to handle multiple sets of resilience constraints.
3. Other than making them members of a resilient group, pools should not require special configuration (e.g., setting lfs=precious) or be prohibited from being backed by tertiary storage (e.g., by requiring replicas to be written as precious).

These three requirements fundamentally distinguish the new system from the old.

Additional New Requirements

1. Allow for the definition of resilience constraints on the basis of **storage units**.
2. Allow for partitioning of copies among pools based on pool characteristics.
   - This is a generalization of the current feature allowing one to exclude multiple copies of the same file from being placed on the same host.
3. Allow for dynamic redefinition of resilience specifications (i.e., without restart of the service).
   - Changes in the composition of pool groups or resilience constraints on a storage unit should, when appropriate, trigger a rescan of the pools implicated in the changes. If the partitioning (2 above) constraints change, this may trigger a redistribution of the files in the storage unit.
4. Handle broken/corrupt replicas by removing and recopying when possible.
5. Raise an alarm for fatal copy failures.
6. Raise an alarm when files having no currently available replicas are discovered.
7. Provide a rich set of admin commands for monitoring and diagnostics, as well as for ad hoc operations to adjust replica state.

Retained Requirements

1. Fulfill resilience specifications for a given file by creating the necessary copies or removing redundant ones. To this end, the following must be implemented:
   a. Handling of add cache location and clear cache location messages.
   b. Handling of changes to pool status (enabled, read-only, offline, dead, etc.).
   c. Periodic scans of the pools in resilient groups in order to maintain replica consistency.
   d. Recovery of replica state if the service itself goes offline and then is restarted.
2. With respect to (1.b), this means that missing copies should be generated when a pool goes offline, and extras may be eliminated when it comes back online.
   – It should be possible, however, to block this behavior temporarily for a given resilient pool.
3. Provide for a best-effort retry of failed copies when the failure is not fatal.

Implementation Goals

1. (Loosely coupled) integration with other dCache services.
   - Use broadcast of the PoolMonitor to track PoolManager state changes.
   - Rely on the full-featured support for p2p provided by the migration module via an independent migration task run in the resilience service itself.
   - Rely on the namespace database (Chimera) via a namespace provider specific to the resilience service.
2. Isolation of resilience operations.
   - A self-contained dCache cell/service.
   - May run in the same domain as other services (provided adequate heap allocation), but can be enabled and disabled without stopping and restarting the entire domain.
3. Should not affect the performance of other services, particularly PnfsManager and PoolManager.
4. Scalability.
   a) Should be capable of handling, in memory, millions of operations.
   b) As the number of operations increases, performance should not degrade due to internal factors (though it may be affected by such things as size of the database, size of the pools, load and network).
      - Benchmarked against an installation of ~100 pools with ~1M files each.

Implementation Goals (cont'd)

5. Fairness and anti-starvation.
   - For a large system with scores of millions of files, a pool scan can queue up quite a number of operations.
     i. Balance work between handling of foreground (newly arriving files) and background (existing files from a pool scan), but never allow the number of running jobs in either to go to 0 if that category is not empty.
     ii. Prefer the availability of at least two replicas; that is, promote to the head of the waiting list operations on files with currently only one replica, and requeue the operation for any additional copies requested.
6. Allow for individual tuning.
   - No different from what we do in general. Defaults have been derived from the benchmark, but properties pertaining to concurrency, timeouts and work limits can be adjusted as needed.
7. Find correct balance between reliability and performance.
   - Recovery mechanisms (such as checkpointing) need not guarantee "losslessness" when this means costly locking; this can be compensated by back-up mechanisms (such as periodic verification done on the pools).

Basic Design

A simplified, high-level view

ResilienceMessageHandler

Listens for **PnfsAddCacheLocation**, **PnfsClearCacheLocation**, and **CorruptFile** messages; internal pool status updates are also routed through this handler.

PnfsOperationHandler

Contains the logic for determining if a pnfsid needs handling, and for selecting and executing the necessary actions.

"Resilience Central": the main locus for tracking operations on files. Contains the state and queueing logic for each pnfsid which needs action.

PnfsOperationMap

BackloggedMsgFile

saved to file when message handling is temporarily disabled

Checkpoint File

contents of the map are written to a checkpoint file periodically

Read Only

Two kinds of information are requested of the namespace: file attributes, and a list of pnfsids with a given location.

NamespaceAccess

Chimera DB

Pool Monitor

PoolManager broadcasts a PoolMonitor refresh every 30 seconds. A change handler compares new and old states of the monitor to see if there are any updates to be made. Some map updates will also trigger scan operations.

PoolInfoMap

Information on pools, pool groups, storage units, along with pool tags, pool cost and current pool status, derived from the PoolMonitor.

Mostly Read

PoolOperationHandler

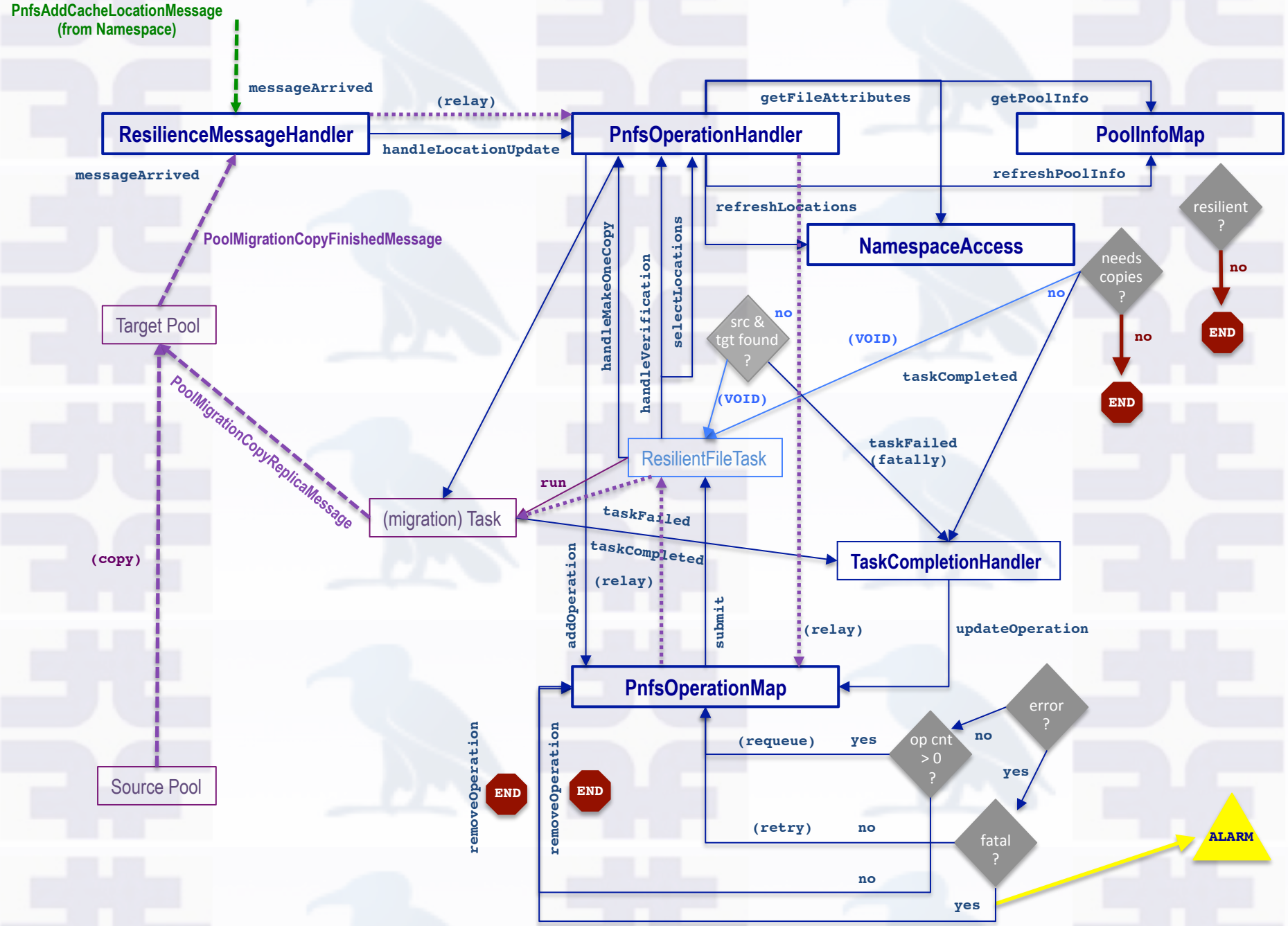
Contains the logic for executing pool scans.

Scans are triggered on pool status changes, periodically, and also by admin command.

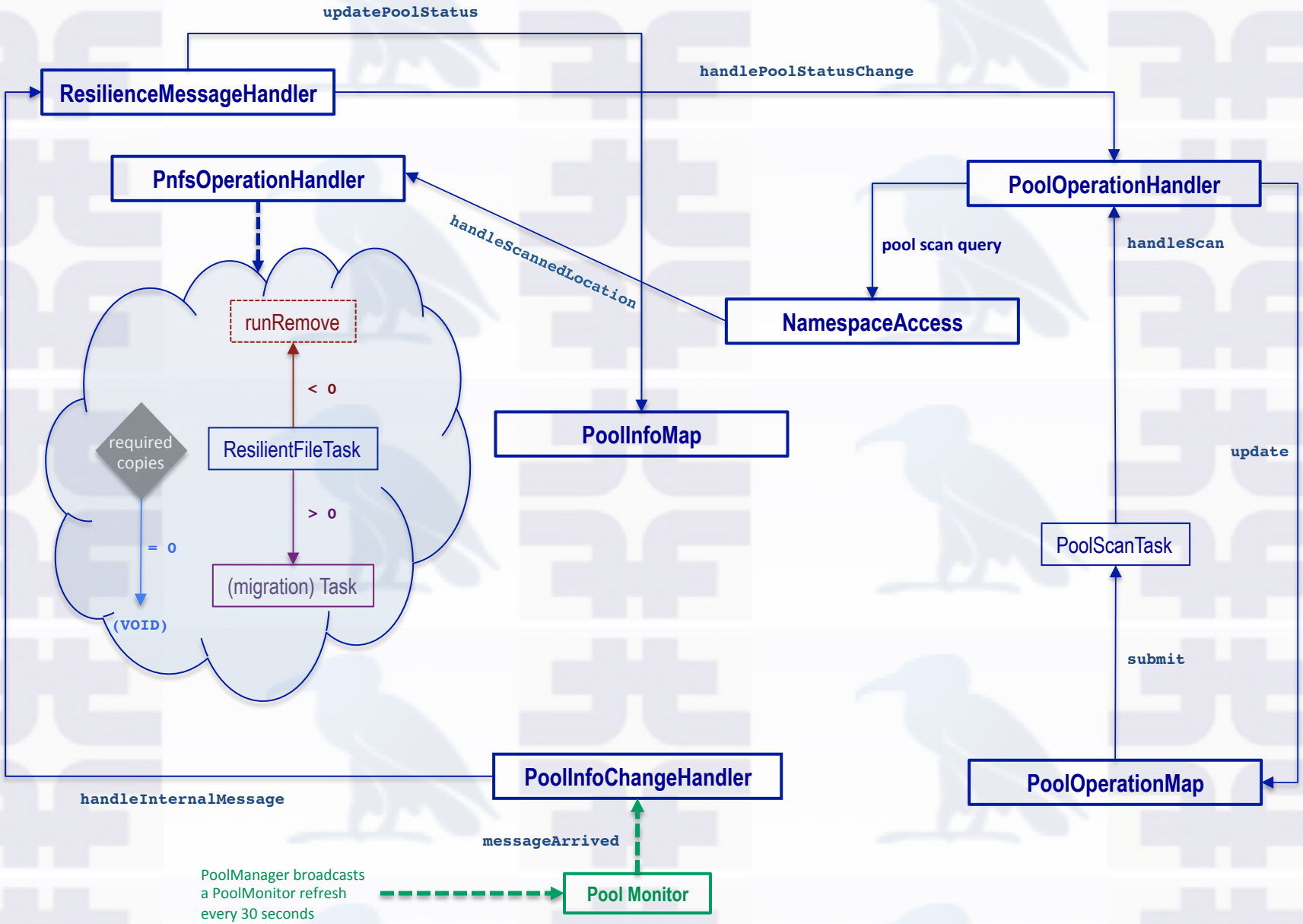
PoolOperationMap

For every resilient pool in the PoolInfoMap an operation record is kept here; contains the state and queueing logic for pool scans.

Example 1: Handling of New File



Example 2: Pool Status Change



Design Remarks: Concurrency

1) While multiple file operations run concurrently (default is 200), operations are serialized against pnfsid (that is, only one operation for a given file is running at a given time). Files requiring more than one copy or remove have each done in sequence.
2) Multiple pool scans can run concurrently, though there are diminishing returns on performance if the number is set too high (default is 5). When separate pool scans request operations on the same pnfsid, the operation count is simply incremented for that file. Since attribute information (including location) and pool information is refreshed for each pass of the operation, the operation will only run as many times as is necessary to fulfill the resilience constraints for that file.
3) An explanation of the various queueing parameters is given in the `resilience.properties` file, should further tuning be necessary. (The defaults are already set so that workers do not block waiting for a database connection.)

How to Activate Resilience

- The resilience service can be run out of the box. All that is required is to include it in some domain.
- Resilience communicates directly with chimera, so **`chimera.db.host`** should also be set explicitly if resilience is not running on the same host as the database.

```
[someDomain/resilience]
chimera.db.host=host-where-chimera-runs
```

Memory Requirements

- While it is possible to run resilience in the same domain as other services, memory requirements for resilience handling are fairly substantial.
- We recommend at least 8GB of JVM heap be allocated, but a safer setting is 16GB. Be sure to allow enough memory for the entire domain.
- If feasible, it is recommended to give resilience its own domain.

Resilience Semantics

- A 'resilient' file is one whose **`AccessLatency`** is ONLINE and whose **storage unit** defines the number of required copies as > 1 (the default).
- Note that **`RetentionPolicy`** is not limited to REPLICA here (that is, one may have CUSTODIAL files which are also given permanent on-disk copies).
- To be resilient, a file must also reside on a pool belonging to a **resilient pool group**.

Setting Up Resilience

To have a fully functioning resilient system, one must do the following:

1. Define one or more resilient pool groups.
2. Define one or more storage units with resilience constraints set (and linked to a resilient group or groups).
3. Create the directories with the necessary tags.

Defining A Resilient Pool Group

To make a pool group resilient, simply add the '-resilient' flag in **`poolmanager.conf`**:

```
psu create pgroup resilient-group -resilient
```

Once a pool group is defined as resilient, it will become "visible" to the resilience service.

- A pool may belong to ***only one resilient group*** (though it can belong to any number of non-resilient groups as well).
- When the resilience service selects a location for an additional copy, it does so from within the resilient pool group of the file's source location.
- Be careful when redefining a pool group by removing its resilient flag. (Note that this is not possible through the admin interface; it requires by-hand modification and reloading of the **`poolmanager.conf`** file. This is a safety precaution.) Doing so will make all files on all pools in that group no longer considered resilient replicas (but remember they are indefinitely "pinned" or sticky, and thus not susceptible to garbage collection).

Defining A Resilient Storage Unit

There are two attributes for a storage unit which pertain to resilience.

1. **required** defines the number of copies files of this unit should receive.
2. **onlyOneCopyPer** refers to pool tags; a comma-delimited list of tag names indicates that copies must be partitioned among pools such that each replica has a distinct value for each of the tags in question. To make sure the replicas get placed on different hosts, for instance, one would include the 'hostname' tag in the layout configuration for each pool in the group, and set this attribute to 'hostname':

```
psu create unit -store test:resilient1@osm
...
psu set storage unit test:resilient1@osm -required=2 -onlyOneCopyPer=hostname
```

Configuration Example

The normal process of linking pools, pool groups and units continues to apply. Thus, to demonstrate the setup for a single resilient pool, pool group and storage unit:

```
psu create unit -store test:resilient1@osm
...
psu set storage unit test:resilient1@osm -required=2 -onlyOneCopyPer=hostname
...
psu create ugroup resilient1-units
psu addto ugroup resilient1-units test:resilient1@osm
...
psu create pool resilient1-pool1
...
psu create pgroup resilient1-pools -resilient
psu addto pgroup resilient1-pools resilient1-pool1
...
psu create link resilient1-link resilient1-units …
...
psu add link resilient1-link resilient1-pools
...
```

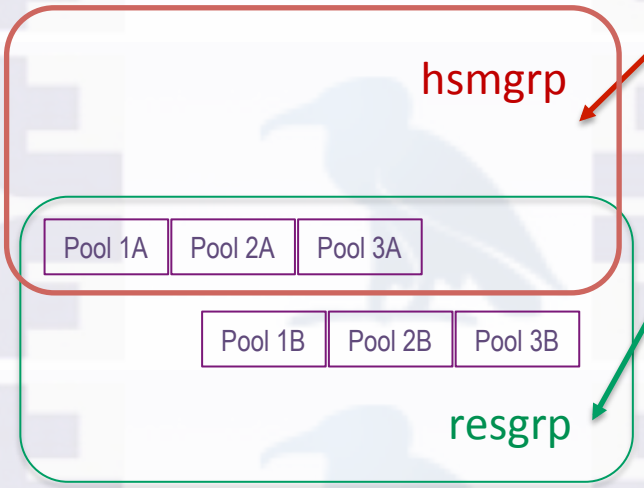
Setting the Directory Tags

The tags to be set in the directories whose files should be treated as resilient, to follow the above configuration, would minimally be:

```
.(tag)(AccessLatency):ONLINE
...
.(tag)(sGroup):resilient1
```

Pool Sharing

It is possible to share pools between a resilient group and a group backed by HSM. The files can be mixed, and the HSM files will not be replicated. Example



hsmunit:

file1: written to directory with tags CUSTODIAL NEARLINE hsmunit

resunit, req = 2:

file2: written to directory with tags REPLICA ONLINE resunit

Because the resilience service checks the AccessLatency attribute, if the file is NEARLINE it is ignored. This pertains to pool scans as well.

Thus 1A, 2A and 3A can have both replicas and cached copies.

Cached Files on Resilient Pools

On resilient pools, any such cached copies are assumed to be from files with AccessLatency NEARLINE. Any ONLINE file which is written to a resilient pool will have the sticky bit owned by 'system' forced there as well.

- How could a cached copy of an ONLINE file without a system sticky bit arrive at a resilient pool?

p2p

  - either an ad hoc p2p via an admin command (***don't do it!***)
  - or replication across links

- Resilience guards against leaving this file cached by forcing its sticky bit, because otherwise this would complicate and render less efficient the counting procedure used to determine if a replica needs additional copies or not.

Recommended Best Practice

1. Rebalancing should rarely be necessary on resilient pools, but if you should decide to rebalance a resilient pool group, be sure to disable resilience. One can do this on a pool-by-pool basis, or by globally disabling resilience:

```
\s poolname[,poolname …] pool suppress resilience on
                        - OR -
\s Resilience disable strict
```

2. Hot replication should be turned off on resilient pools.

Resilience Home

On the host where resilience is running, you will see several files in the resilience home directory (**/var/lib/dcache/resilience** by default)

- **pnfs-operation-map**
- **pnfs-backlogged-messages**
- **pnfs-operation-statistics**
- **pnfs-operation-statistics-task-{datetime}**
- **{poolname}-inaccessible-files**

Checkpointing & Message Backlog

pnfs-operation-map

is a checkpointed image of the main operation table. By default, this snapshot is written (in simple text format) every minute. It is a heuristic for immediately reprocessing incomplete operations in case of a domain crash and restart. Note that it is an approximation, so not all file operations in progress at that time may have actually been saved. The fallback for this is that the periodic pool scan should eventually detect any missing replicas not captured at restart/reload.

pnfs-backlogged-messages

It is possible to enable and disable message handling temporarily inside resilience. There is also a short delay at startup which may also accumulate a backlog. Messages which arrive during this period are written out to this file, which is then read back in and deleted during (re)initialization.

Inaccessible Files

`{poolname}-inaccessible-files`

When files subject to replication are discovered to have no currently available copies (all its locations are not readable), its pnfsid is written to this file. This action is also associated with an alarm, but no attempt to rectify the situation is made (that is left to the administrator for the moment).

Statistics

- `pnfs-operation-statistics`
- `pnfs-operation-statistics-task-{datetime}`

One can optionally turn on the recording of statistical data. Two kinds of files are produced. The first is a general overview of performance with respect to pnfs messages and operations. The second are detailed, task-by-task records and timings which are logged to a file which rolls over every hour.

Actions Available via Admin Command

1. Enable and disable resilience message handling.
2. Enable and disable all of resilience without having to stop the domain.
3. Print to a file (in /var/lib/dcache/resilience) the pnfsids and their replica counts for a given pool.
4. Run a job to adjust a given pnfsid (i.e., make required copies or remove unnecessary ones).
5. Reset properties controlling the internal handlers and maps (such as sweep intervals, grace periods, etc.).
6. List the current pnfs operations, filtered by attributes or state; or just output the count for the given filter.
7. List pool information derived from the pool monitor.
8. List pool operations, filtered by attributes or state.
9. Cancel pnfs operations using a filter similar to the list filter.
10. Cancel pool operations using a filter similar to the list filter.
11. Initiate forced scans of one or more pools.
12. Set the status of a pool operation to exclude or include it from scanning.
13. Display diagnostic information, including number of messages received, number of operations completed or failed, and their rates.
14. Display detailed transfer information by pool and type (copy/remove), with the pool as source and target.
15. Enable or disable the collection of statistics to files in resilience home.
16. Display the contents of the diagnostic history file.
17. List the most recent pnfsid operations which have completed and are now no longer active (a 'history' buffer). Do the same for the most recent terminally failed operations.

The new resilience service will be included in golden release 2.16.

The old Replica Manager will be deprecated, but still available.

Feedback is welcome, especially from those of you who make use of this feature in production.

Script for Demonstrating Admin Commands (1)

ACTION	TEXT
`\c Resilience`	Connect to the resilience service (using its exported well-known cell name).
`\h`	The commands specific to resilience are: counts, diag, diag history, disable, enable, pnfs cancel, pnfs list, pool group info, pool cancel, pool info, pool list, pool scan
`diag`	diag, or diagnostics, prints out information about the system such as uptime, last sweep times of pnfs operations, checkpointing, and cumulative totals for messages and actions. The rates associated with these are sampled based on the checkpointing interval.

`diag .*`
`diag dmsdca17`
`diag dmsdca17-1$`
	adding a regular expression argument will display a detailed list of transfers by pool. The sizes represent the total bytes received as target.
	regex is greedy with no delimiters
	this applies to all the commands that take regex arguments

`diag history` `diag history -enable`	diag history displays the operations statistics file; statistics collection is disabled by default We will turn it on to show the output during our small test.
`pool info` `pool info dmsdca17`	Pool Monitor information is reparsed into structures useful to the resilience service. Basic pool information can be seen using this command (which also takes a regular expression argument)
`pool ls` `\h pool ls`	For each resilient pool, resilience stores an operation marker; ls displays these. This command, along with pool cancel and pool scan, accepts a number of attribute filters
	We are going to use the -state filter (WAITING, RUNNING) to check up on a pool operation momentarily
	By way of demonstration, we will now do the following: - disable a pool - examine the results - re-enable the pool before its scan completes - examine the results
	for lack of time, I will not show here the behavior of resilience when the pool down scan completes (to wit: a down pool is scanned only once, and will not be rescanned automatically until it comes back on line)

Script for Demonstrating Admin Commands (2)

```
pool ctrl info
```

Before we do this, let's have a look at the pool scan settings.

This command controls various timeout and queueing properties.

We won't demonstrate all of these. What I want to focus on is the grace period.
These exist because it may be desirable not to take immediate action when a pool goes down or comes back on line.
(There is also a way to disable resilience handling on a pool by using a pool command, which I will show shortly.)

```
pool ctrl reset -down=10 -unit=SECONDS
pool ctrl reset -restart=10 -unit=SECONDS
pool ctrl info
```

As you can see, both down and restart have a large delay (1 DAY). Let's set these to 10 seconds.

```
\s dmsdca17-1.1 pool disable -strict
```

Now we can observe the action taken when a pool goes down.

Let us issue a disable command to a pool.

```
pool ls dmsdca17-1.1
```

note it is in the WAITING state
note also that the pool sweep period is set to 3 minutes. Rather than waiting for the next sweep, let's interrupt the sweeper

```
pool ctrl run
```

and make it run immediately.

```
pool ls dmsdca17-1.1
```

now we can see the operation is running. We can also see the total number of files it needs to handle and the percentage done.

```
pnfs ls
```

If you wish to examine (or cancel) individual file operations, use the pnfs version of the pool commands we have seen
these also have filtering attributes

```
pnfs ls $@ -state=RUNNING
pnfs ls $@ -state=WAITING
```

If you just want to see a count of operations, with or without filter values, append a $ for the number of operations, and an '@' for the counts broken down by pool or origin. Note that count >= operations, because some pnfsids may require the operation to be repeated.

```
history
history errors
```

We can also see operations which have recently completed. (The buffer size is 5000 by default)
If there are errors, these can be viewed apart

Script for Demonstrating Admin Commands (3)

```
\s dmsdca17-1.1 pool enable
```

Now let us re-enable the pool

```
pnfs ls
history
pool ls dmsdca17-1.1
pool ctrl run
pool ls dmsdca17-1.1
```

Notice there are no more operations
History shows they have been canceled.
The pool operation has been restarted fresh. Let's force a run

```
history
```

Now we should begin seeing remove operations, since there were extra copies created by the pool down which are no longer necessary

```
\s dmsdca16-1.1 pool suppress resilience on
\s dmsdca16-1.1 pool disable -strict
```

We can suppress resilience on a pool. This flag is part of the setup; doing save will make it survive a pool restart.
If we then disable the pool,

```
pool ls dmsdca16-1.1
\s dmsdca16-1.1 pool suppress resilience off
```

the status of the pool is inactive, and the job state is IDLE, since no action is taken.
turning off suppression will immediately notify resilience

```
pool ls dmsdca16-1.1
\s dmsdca16-1.1 pool enable
pool ls dmsdca16-1.1
```

and the pool operation will go to waiting, as usual, to process the pool as DOWN
or as enabled

```
pool cancel dmsdca16-1.1 -includeChildren
```

let's cancel it. The include children flag here is not strictly necessary, as the operation has not begun to run, but the flag indicates that any unfinished file operations should also be canceled.

```
diag history
```

here we can see the operation statistics recorded since we enabled this
(to see the detailed task-by-task statistics, you must log on and examine the file in /var/lib/dcache/resilience)

```
pool group info -showUnits -verify
```

One last command I'd like to mention is pool group info
This provides a way to see which storage units are linked to a resilient group, and to check that the pools in the group are sufficient to meet the requirements for all those units