From the centralized Resilience service to "resilience" on the pools

The following are some remarks on a possible redesign of Resilience to make it fully integrated into dCache.

I am skipping any intermediate versions of the transition in order to conceptualize how I see a system in which there is no "Resilience" per se, but simply the handling of a file policy.

I am working from the assumption that there should be no special service, no special categorization of pool groups or pools, and no special status given to files requiring permanent disk copies. All we are interested in, *grosso modo*, is (a) whether a file lives on tape and/or disk, (b) if it lives on disk, how many copies of it should have some well-defined lifetime, (c) how those copies should be distributed across pools according to other requirements (currently the pool tags).

1. Modifying the pools to handle policy

In order to make each pool "responsible" for the policy of the replicas it contains, I suggest that (at least) two components need to be added.

* 1. *Policy Diff Handler.* This component would be similar to what exists in Resilience currently vis-à-vis PoolMonitor changes. It would subscribe to a topic to receive updates from the «Policy Engine» concerning changes, and it would update its own internal copy of the policy map in response to it.
  2. *Group Status Handler*. This component would possess knowledge of which group(s) the given pool belongs to. It will also need to access the internal policy map. More on its responsibilities follows below.



1. Handling New Files

After a file is written to a pool, the pool needs to trigger a task which does the following:

1. Looks up the file’s policy;
2. Establishes how to copy, lifetime and owner of any pins, etc.;
3. Selects other pools in an appropriate pool group;
4. Launches (concurrent) migration jobs on those target pools to use the new file as source.

***Remarks***

(1) is currently handled in terms of storage unit; the file attributes (from Chimera) are compared to the mapping of storage unit to requirements. Certainly the file attributes will be extended in some way to convey the new policy (an id, perhaps?), and that info may be directly sent to the pool when the pool initially writes the file. Similarly, location information could be sent directly (also as file attributes). Thus on an initial write, the pool may be able to skip further verification (against the namespace). This could of course introduce the possibility of races in terms of location count. Are we OK with allowing migration module idempotency to handle this? That is, there may be redundant launching of migration jobs, but these should quit when they find that the required copies are already present ...

(2) We need not limit ourselves to files requiring permanent replicas. One can make copies for a given lifetime, either owned by the system or by the user, as well.

(3) This would entail each pool knowing what group(s) it belongs to. The Group Status Handler would contain these mappings. Note also that if we remove the distinction between resilient and normal pool groups (as it seems it would make sense to do), then we would simply need to extend the pool selection algorithm to include a way of selecting the pool group from which to choose target pools for replication, should the pool belong to multiple groups. The important thing is to maintain consistency.

1. Pool Status Changes

When a pool goes offline (DOWN) or comes back online (UP), the pool's files must be checked to see if there are adequate or excessive copies, and appropriate action taken.

The problem is: who should be responsible for this? Currently, it is a central service (Resilience). In the redesign, we need something like the following:

1. An election process establishes which pool is the group leader;
2. The Group Status Handler of each pool knows who the leader is;
3. The Group Status Handler of each pool subscribes to PoolMonitor updates (or whatever the equivalent of this will be; one might even wish to decentralize this and simply pass a heartbeat message around among pools in the pool group ...);
4. If there is a pool status change of a member pool, and the pool receiving the notification group leader, it takes the action described above;
5. If there is a pool status DOWN of a member pool, and that pool is the leader, a new election takes place, and the new leader proceeds as above;
6. Otherwise, the pool’s Group Status Handler does nothing.

Response to a pool status change involves acquiring a list of files on the relevant pool. This is done currently using a Chimera query. It cannot (?) be done on a DOWN pool using the pool repository. So the Group Status Handler needs access to Chimera and must run a similar query as before. Hence there is still a need for a "master" component (though now one per pool group, rather than for the entire system). It may be desirable to "round-robin" this responsibility during idle times (when the group leader is not involved in scanning), if there is a concern that the load/activity involved may overburden a particular node or JVM.

Note also that the procedure for counting replicas becomes more complex/expensive because we now need to check the pool repository lifetimes and owners, for instance, for each replica.

1. Policy Status Changes

Just as the PoolManager currently does, the «Policy Engine» should publish state periodically on a special topic that can be subscribed to. The Policy Diff Handler on the pool would receive these messages, and like the current Resilience service, update its internal policy map when there are changes.

Some of these changes may entail the need to make or remove replicas, or redistribute them, change their lifetime, etc. The current procedure for handling this is as follows: through the PSU, Resilience looks up (by following links) which pools could have files with the storage unit which has changed, and those pools are scanned (as above). Since this only involves UP pools (DOWN pools will be handled by the group leader), it might at first seem to make more sense for each of those pools to do its own work; however, we are here implying the launch of a migration job for each of its files (unless each pool also verifies each file against Chimera ... not a good idea), with the added consequence that a major proportion of these may be redundant, since pools with overlapping files may be involved. It is thus probably advisable to do this in the Group Status Handler of the leader as well, imitating once again the current Resilience service. There will still be somewhat better (at most linear) scaling because each group has its own group handler. Having the group leader take care of this also makes more sense with respect to removal, because one can serialize (as with the current Resilience engine) on *pnfsid* rather than trying to build into removal a rather complicated synchronization to handle potentially concurrent removes of replicas of the same file.

1. Periodic verification

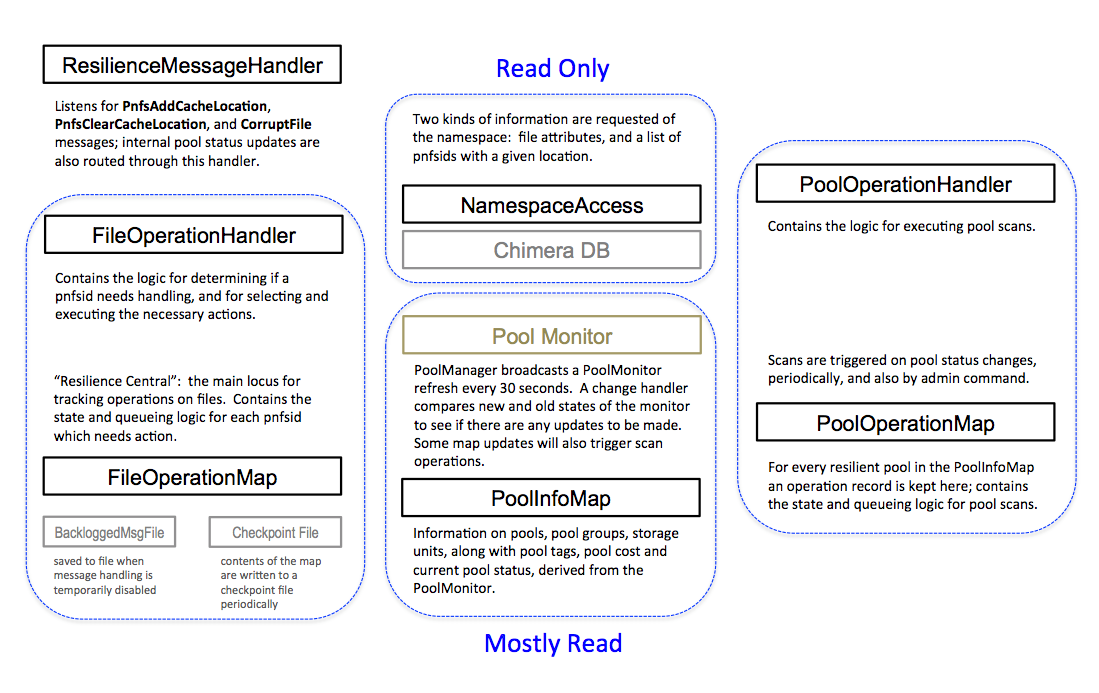
Resilience currently does this on a pool-by-pool basis, running a query on Chimera and launching the necessary copies/removes. As with the policy status change handling, there will exist the same trade-offs in trying to decentralize this work, so this too probably belongs in the Group Status Handler of the pool group leader.

A few more remarks on the Group Status Handler. Status changes could potentially be processed serially by pool. Since we are handling only pools in a single group, there would still be concurrency when different groups were involved. Handling only one pool at a time greatly simplifies the logic and lowers the upper bound requirements on memory.

The isolation of scanning to within a pool group might at first seem to raise the question of overlapping pool scans not present in the current Resilience setup. For example, if a pool goes down, which group is responsible for the scanning? But the answer has to be: all groups it belongs to. But then that would indeed mean that two or more scans of the same pool would be running simultaneously, even if pool scans are serialized within a given Group Status Handler. Nevertheless, if we are consistent in the way that the target pool group is chosen for a particular file, then any given file should only belong to one of those groups. In order to make sure that all files on a down pool are covered, the Group Status Handler (responsible for only its own group) for each group must be run, but there should be no overlapping *pnfsids* in this case (there is an implicit equivalence relation on files over groups).

1. Reusability of current code

To remind you of what the current architecture of the service looks like:



The central maps and queue logic for handling file operations and pool scans can be passed to the Group Status Handler. The PoolInfoMap and PoolDiffHandler can be adapted to become the Policy Diff Handler, though this may entail fundamental data structure changes. The Group Status Handler will need to access the internal policy map. Scanning should entail either serialization on pnfsid, or perhaps coarser grained serialization on the pool, with respect to any given pool group.

What will change significantly is the verification/copy handling procedure for new files, since this task will be run by each pool. There will be some additional functionality adjustments to deal with file lifetime requirements (sticky bit handling). Similar to what exists in Resilience, a distinction will need to be made (on the basis of something like a session id) between writes that originated from a door and writes initiated from within the Group Status Handler, so that replicas produced via the migration jobs not trigger additional migration jobs.

Another change will be the loss of central administrative control: diagnosis, monitoring, etc., will have to be on a pool-group-by-pool-group basis. One could perhaps retain the admin component and run it in some chosen domain, so that under the covers it would collect information from the pool groups; but given the way the shell works, it would perhaps be simpler to come up with a way of locating the pool group handlers that are the leaders, and then just allow the current Resilience commands to be invoked on the leader pools (e.g., "\s leader1,leader2,leader3 diag").

1. Possible intermediate implementations

Given the fact that scanning in general seems best to handle through some master component, it may be possible to implement just the new cache location part (the new file writes) on the pools, and have the current Resilience service limited only to the processing of status changes. The second stage would then be to move this functionality onto the pools as well.

1. Summary remarks

While it makes sense to decentralize the work of maintaining the correct number of copies and to associate this logic directly with the pools, there is still a need for some of this activity to be centralized in a single "master" component. However, if the system is comprised of multiple pool groups, allow each *group* to manage itself seems feasible and may offer greater scalability as well.

The clear disadvantage of placing this activity on the pools is that provision must be made on each one of them for their group handler to be elected leader, meaning increased memory allocation across the board.

Albert L. Rossi

Last modified: Wednesday, February 8, 2017