Replica Manager (work in progress)

Albert L. Rossi

Fermi National Accelerator Laboratory

Resilience

- → ***dCache Book*, Chapter II. 6.**
  - http://www.dcache.org/manuals/Book-2.7/config/cf-repman-fhs.shtml

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

  - for higher security and/or availability in absence of tertiary file system:
    - Uses p2p to guarantee number of copies of a file is at least 2.
    - If four or more replicas exist, some of them will be deleted.

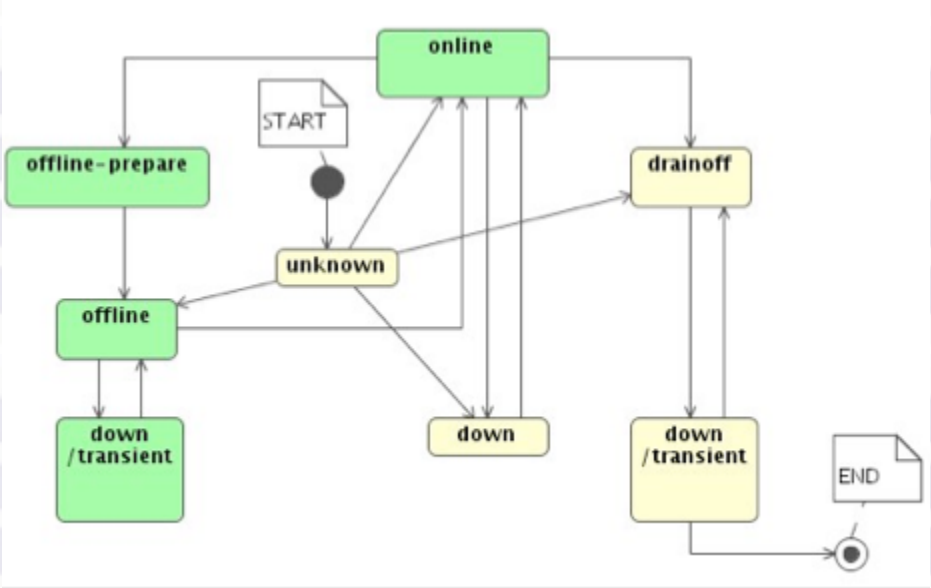
  - hybrid mode (resilient pool group, non-resilient groups for HSM, etc.)

Current Replica Service

- No significant modification since 2007.
  - Maintains a rather heavyweight set of database tables (written before move from PNFS to Chimera).
- The basis for replication is established by the pool the file ends up on; if it is in the resilient group, it gets replicated, regardless of the storage information associated with it; hence replication must ultimately be controlled by the way links direct files to pool groups.
  - Is this how it should work? (what about tags? storage class?)
  - If it is, is the current behavior consistent?
- 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. This entails hacking broadcast.batch as only one Replica Manager is supported.
  - Replica range is fixed to 2 <= n <= 3 for all pools in the group.

Pool States & Replication



1: Initial state, 2<= N <= 3

- All pools are online

	Pool 1	Pool 2	Pool 3	Pool 4	Pool 5	Count
File A	A	A				2
File B	B		B			2
File C		C	C			2
File D			D	D		2
	online	online	online	online	online	

2: Pools 1 and 2 went down

- Can't access File A; replicate B and C

	Pool 1	Pool 2	Pool 3	Pool 4	Pool 5	Count
File A	A	A				0
File B	B		B → *B*			1
File C		C	C →		*C*	1
File D			D	D		2
	down	down	online	online	online	

2: Set pools 1 and 2 to drainoff

- File A extracted from pool 1

	Pool 1	Pool 2	Pool 3	Pool 4	Pool 5	Count
File A	A	A →			*A*	0
File B	B		B → *B*			1
File C		C	C →		*C*	1
File D			D	D		2
	drainoff	drainoff	online	online	online	

2: set pool pool_1 offline

- Temporarily take pool out, no replication

	Pool 1	Pool 2	Pool 3	Pool 4	Pool 5	Count
File A	A	A				2
File B	B		B			2
File C		C	C			2
File D			D	D		2
	offline	online	online	online	online	

Tags & Pools

How does/should replication take into consideration the relationship between the following?

1. Retention Policy (REPLICA, CUSTODIAL, OUTPUT)
2. Access Latency (NEARLINE, ONLINE)
3. Large File Store pool settings (*none*, *precious*, *volatile*)

When does it make sense, for instance, to replicate files with CUSTODIAL retention policy? What about those with NEARLINE access latency? (As it stands, dCache relies on the admin to set up pools and tags in a way that makes sense.)

Tags & Pools

An experiment to test for consistency the current handling of replication according to these tags and large file store type.

With a resilient group of two pools, the following results were obtained:

POOLS

- 08 = v-dmsdca08-1
- 10 = v-dmsdca10-1

DATA

- SI = tags given by PnfsManager storageinfoof
- CI = pool copies given by PnfsManager cacheinfoof
- ST = Entry State (Cached C, Precious P, Sticky X)

	REPLICA ONLINE	REPLICA NONE	CUSTODIAL ONLINE	CUSTODIAL NEARLINE
lfs=none	run-none-140108Jan001389211254	run-none-140108Jan011389211262	run-none-140108Jan011389211269	run-none-140108Jan011389211276
	0000BA9786E13F7543199D35D63E680D3255	0000D39BC75EF18645CC96A911A16EC6A74C	0000D0310AF8077A4D0EA7C0E6F75C2D4C9F	00006F7E9ED2AF144448BBC3A54A9B4ED732
	SI: REPLICA ONLINE	SI: REPLICA NEARLINE	SI: CUSTODIAL ONLINE	SI: CUSTODIAL NEARLINE
	CI: 08, 10	CI: 10, 08	CI: 10, 08	CI: 08, 10
	ST: CX, P	ST: C, P	ST: PX, P	ST: P, P
lfs=precious	run-precious-140108Jan591389211154	run-precious-140108Jan591389211162	run-precious-140108Jan591389211169	run-precious-140108Jan591389211177
	00009F4EAB7582C94C11A4CB852D2B43EF4D	0000CE905AB86E1D46AD8F2213F8F6D68AC8	0000F58ED7CE8B0941619741B288A514AB02	0000220B23F5AFDE4C1B8419EC877D27A03A
	SI: REPLICA ONLINE	SI: REPLICA NEARLINE	SI: CUSTODIAL ONLINE	SI: CUSTODIAL NEARLINE
	CI: 10, 08	CI: 10, 08	CI: 10, 08	CI: 08, 10
	ST: CX, P	ST: C, P	ST: PX, P	ST: P, P
lfs=volatile	run-volatile-140108Jan021389211328	run-volatile-140108Jan021389211336	run-volatile-140108Jan021389211344	run-volatile-140108Jan021389211351
	000046D10A6B303C4BD695A51CDE24DDA5F8	0000BCDCF5E60AF045D99B3584A174BD3169	0000CE50A6EAA0924AC683E898769236BF8B	0000BADC9AF020404DCAA3A9B551BBDE3607
	SI: REPLICA ONLINE	SI: REPLICA NEARLINE	SI: CUSTODIAL ONLINE	SI: CUSTODIAL NEARLINE
	CI: 10, 08	CI: 10, 08	CI: 10, 08	CI: 08, 10
	ST: C, P	ST: C, P	ST: C, P	ST: C, P

Tags & Pools

Summary

1. Regardless of retention policy and access latency tags, files are indeed replicated.
2. REPLICA NONE is changed to REPLICA NEARLINE; this combination is somewhat non-sensical, but attempts to specify a temporary copy *without using* ***lfs***.
3. Coming onto the source pool, the file is marked *cached* unless
   a. Retention Policy is CUSTODIAL and Large File Store is not *volatile*;
   b. If CUSTODIAL + ONLINE, original is *precious* + *sticky*.

   ***In other words*:**
      i. REPLICA | volatile => C, else => P
      ii. ONLINE & not volatile => X

   (*This is just the way dCache works*.)
4. But the replicated copy is currently marked *precious* (P).

Regardless of replication policy concerning CUSTODIAL, NEARLINE or precious files, it would make more sense that the resulting replica be CACHED+STICKY.

User-Facing Changes

1. Provide flexibility in defining how replication is handled on the basis of:
   a. Pool Group
   b. Storage Unit (*overrides pool group constraints*)
      - defined on basis of ***storage class***, e.g.

```
psu create unit -store <storage class tag>
```

poolmanager.conf

```
psu set pgroup <group name> [-minreplicas=<integer>] [-maxreplicas=<integer>] \
                            [sameHostEnabled=<true|false>]


psu set storage unit <unit name> [-minreplicas=<integer>] [-maxreplicas=<integer>] \
                                 [sameHostEnabled=<true|false>]
```

NB. There is a tacit assumption that replicating pools must be partitioned by group (that is, any pool in a replicating group may not belong to another replicating pool group).

User-Facing Changes

2. Retain some current admin commands, and add several new ones.

current

set pool <pool><state>	(for controlling in particular drain, offline settings)
show pool <pool>	(show pool state)
ls unique <pool>	(show pnfsids unique to this pool)
exclude <pnfsId>	(do not replicate this particular file)
release <pnfsId>	(allow replication for this particular file)

new

replicate <pnfisd> <pool>	(a single file using a pool as source)
reduce <pnfisd> <pool group>	(a single file to minimum copies)
ls unavailable <pool>	(show pnfsids on this pool with no replicas on *active* pools)
statistics	(current and total counts for requests, messages and queues)
dump queues	(current content of all the queue data)
next scan	(date and time the next pool scan is scheduled to run)
run scan	(force the pool scan to run immediately)

User-Facing Changes

3. Some new properties

replicamanager.limits.pool-scan-initial-wait=1
replicamanager.limits.pool-scan-initial-wait.unit=MINUTES

replicamanager.limits.pool-scan-period=12
replicamanager.limits.pool-scan-period.unit=HOURS

replicamanager.limits.status-workers-per-poolgroup=2

replicamanager.limits.replica-workers-per-poolgroup=10

replicamanager.limits.wait-queue-capacity=10000

replicamanager.requests.use-greedy-limits=true

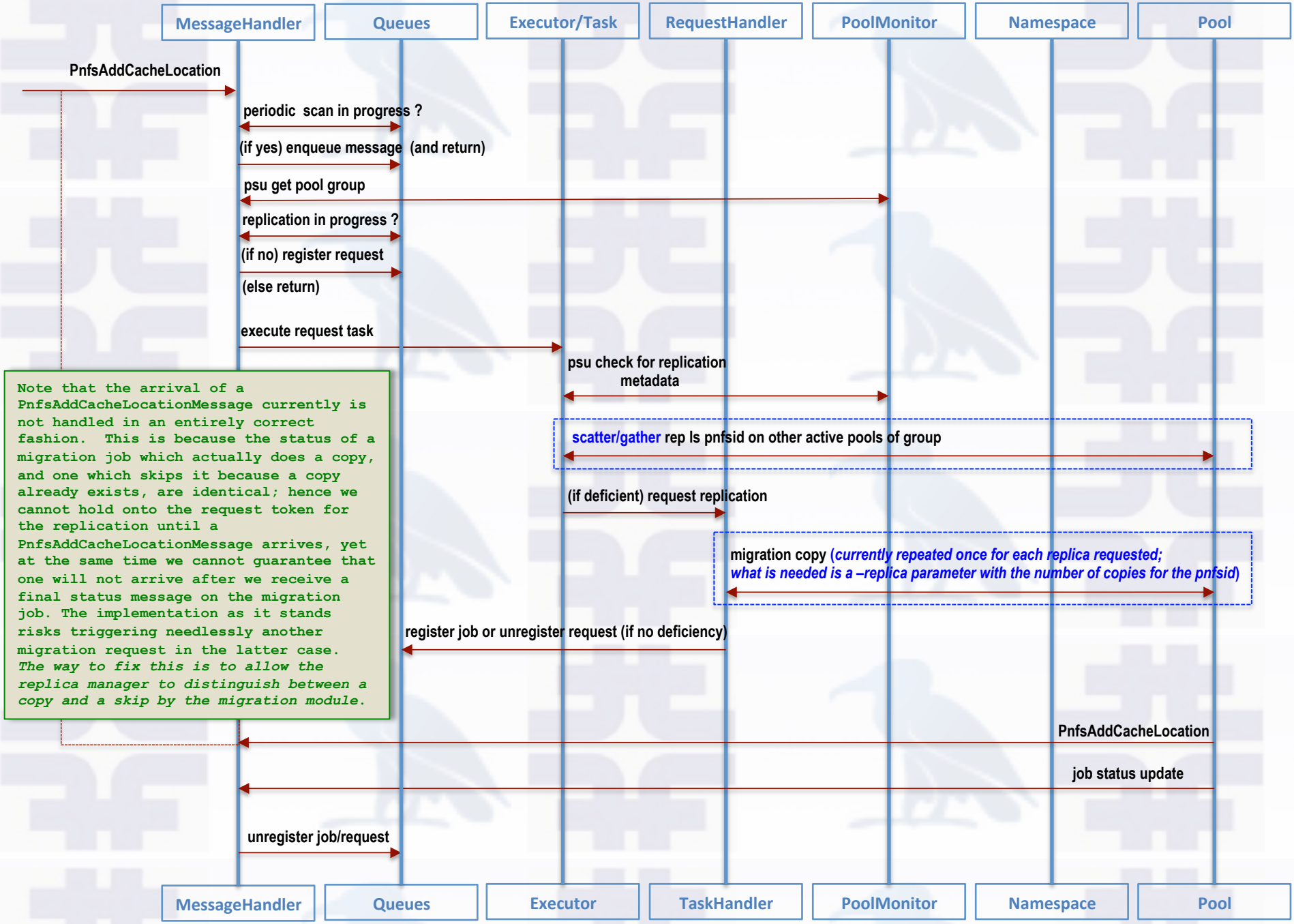
Implementation Goals

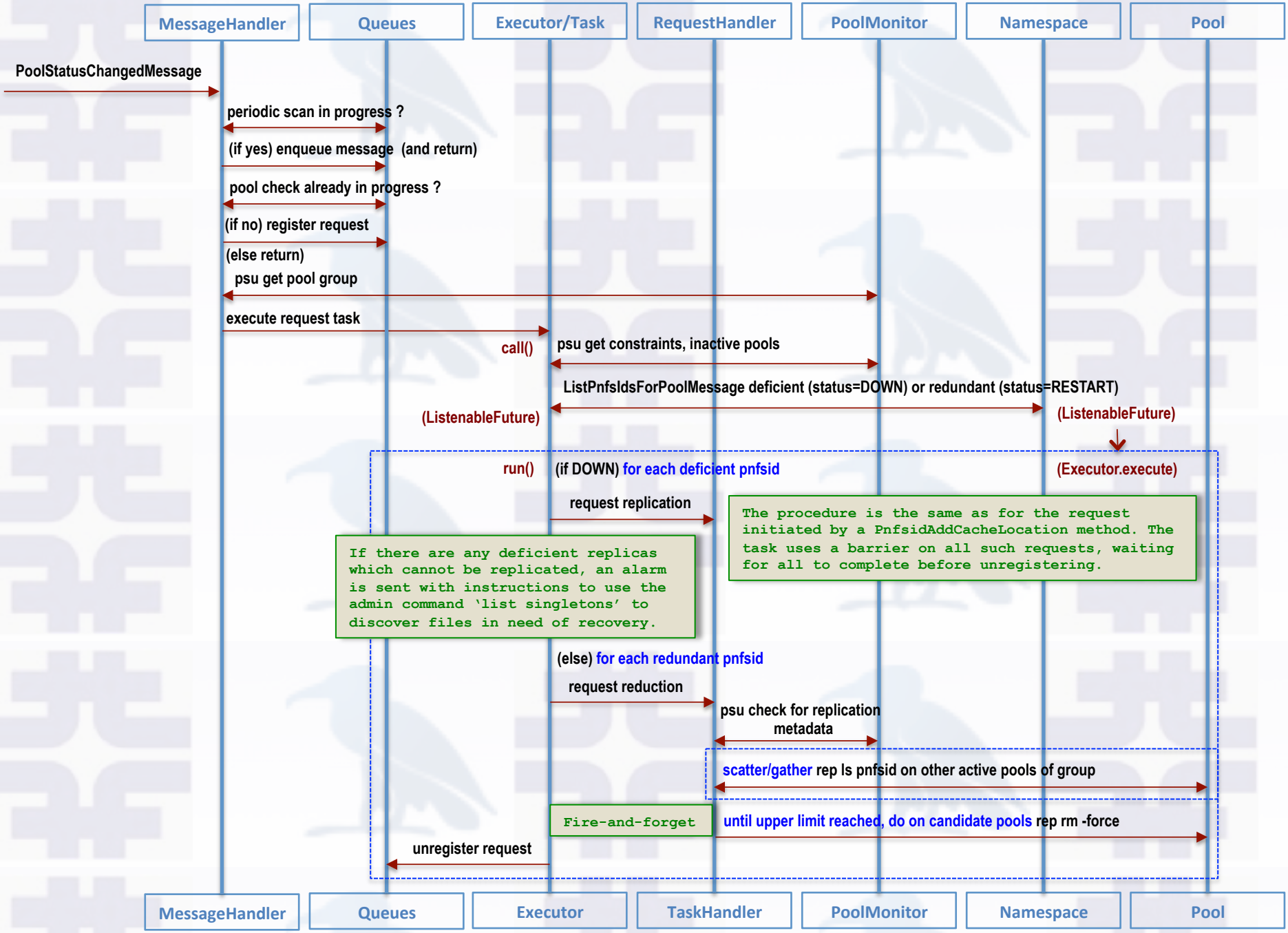
- Make use of existing services and modules (*Chimera*, *Migration Module*) to do the heavy lifting.
- Make persistence more limited and lightweight (eliminate need for *rdbms* and unnecessary replication of stored data).
- Bring code up to date to use modern libraries and OO design praxis.

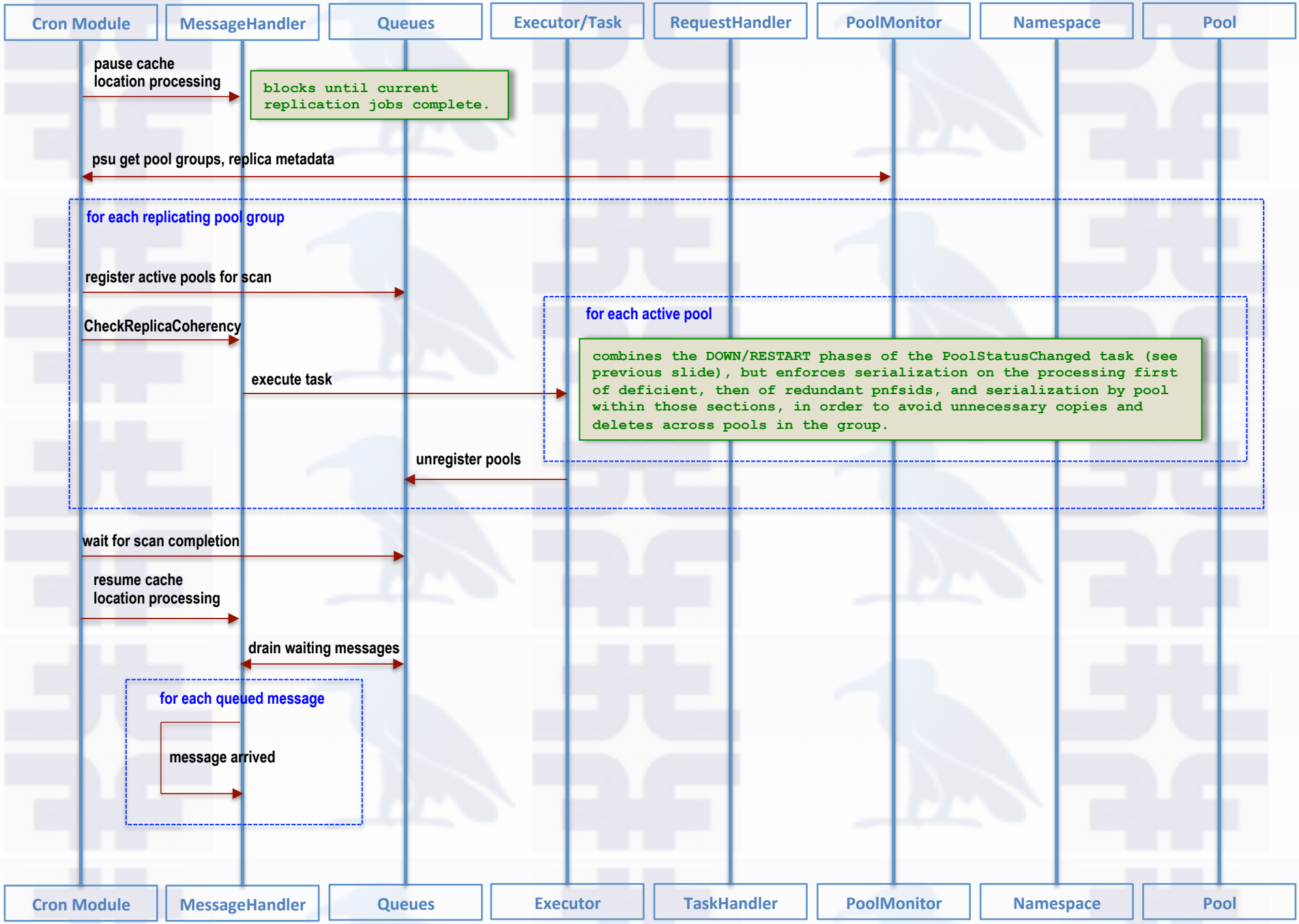
A partial prototype is under review, will go through several more iterations before inclusion in release.

2.11?

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


MessageHandler
Queues
Executor/Task
RequestHandler
PoolMonitor
Namespace
Pool
PnfsAddCacheLocation
periodic scan in progress ?
(if yes) enqueue message (and return)
psu get pool group
replication in progress ?
(if no) register request
(else return)
execute request task
psu check for replication metadata
scatter/gather rep ls pnfsid on other active pools of group
(if deficient) request replication
migration copy (currently repeated once for each replica requested; what is needed is a –replica parameter with the number of copies for the pnfsid)
register job or unregister request (if no deficiency)
PnfsAddCacheLocation
job status update
unregister job/request
Note that the arrival of a PnfsAddCacheLocationMessage currently is not handled in an entirely correct fashion. This is because the status of a migration job which actually does a copy, and one which skips it because a copy already exists, are identical; hence we cannot hold onto the request token for the replication until a PnfsAddCacheLocationMessage arrives, yet at the same time we cannot guarantee that one will not arrive after we receive a final status message on the migration job. The implementation as it stands risks triggering needlessly another migration request in the latter case. The way to fix this is to allow the replica manager to distinguish between a copy and a skip by the migration module.
MessageHandler
Queues
Executor
TaskHandler
PoolMonitor
Namespace
Pool


MessageHandler
Queues
Executor/Task
RequestHandler
PoolMonitor
Namespace
Pool
PoolStatusChangedMessage
periodic scan in progress ?
(if yes) enqueue message (and return)
pool check already in progress ?
(if no) register request
(else return)
psu get pool group
execute request task
call()
psu get constraints, inactive pools
ListPnfsIdsForPoolMessage deficient (status=DOWN) or redundant (status=RESTART)
(ListenableFuture)
(ListenableFuture)
run()
(if DOWN) for each deficient pnfsid
(Executor.execute)
request replication
The procedure is the same as for the request initiated by a PnfsidAddCacheLocation method. The task uses a barrier on all such requests, waiting for all to complete before unregistering.
If there are any deficient replicas which cannot be replicated, an alarm is sent with instructions to use the admin command 'list singletons' to discover files in need of recovery.
(else) for each redundant pnfsid
request reduction
psu check for replication metadata
scatter/gather rep ls pnfsid on other active pools of group
Fire-and-forget
until upper limit reached, do on candidate pools rep rm -force
unregister request
MessageHandler
Queues
Executor
TaskHandler
PoolMonitor
Namespace
Pool


Cron Module
MessageHandler
Queues
Executor/Task
RequestHandler
PoolMonitor
Namespace
Pool
pause cache location processing
blocks until current replication jobs complete.
psu get pool groups, replica metadata
for each replicating pool group
register active pools for scan
CheckReplicaCoherency
for each active pool
execute task
combines the DOWN/RESTART phases of the PoolStatusChanged task (see previous slide), but enforces serialization on the processing first of deficient, then of redundant pnfsids, and serialization by pool within those sections, in order to avoid unnecessary copies and deletes across pools in the group.
unregister pools
wait for scan completion
resume cache location processing
drain waiting messages
for each queued message
message arrived
Cron Module
MessageHandler
Queues
Executor
TaskHandler
PoolMonitor
Namespace
Pool

Addendum: Why Replica Manager should interact with the Migration Module and not the Pool Manager

- Pool Manager manages *transfers.* Its primary job is to do pool selection. This is designed to respond to client read requests by initiating hot spot replication, staging from files, failing on load, dealing with the aging of files, link fallback etc. Different thresholds can be defined for it which are irrelevant to the replication of a file. The Migration Module on the other hand was designed to operate independently of the Pool Manager, precisely because the task of internally moving files is different from clients reading files.

- Pool Manager queues reads to same file, but not different files. Replica Manager should handle all replication requests via queuing/throttling, i.e., allow only X number of replications of different pnfsids proceeding concurrently.