FERMILAB-SLIDES-21-025-SCD
This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

dCache: From Resilience to QoS

Albert L. Rossi (for the dCache collaboration)
Fermi National Accelerator Laboratory
May 18, 2021

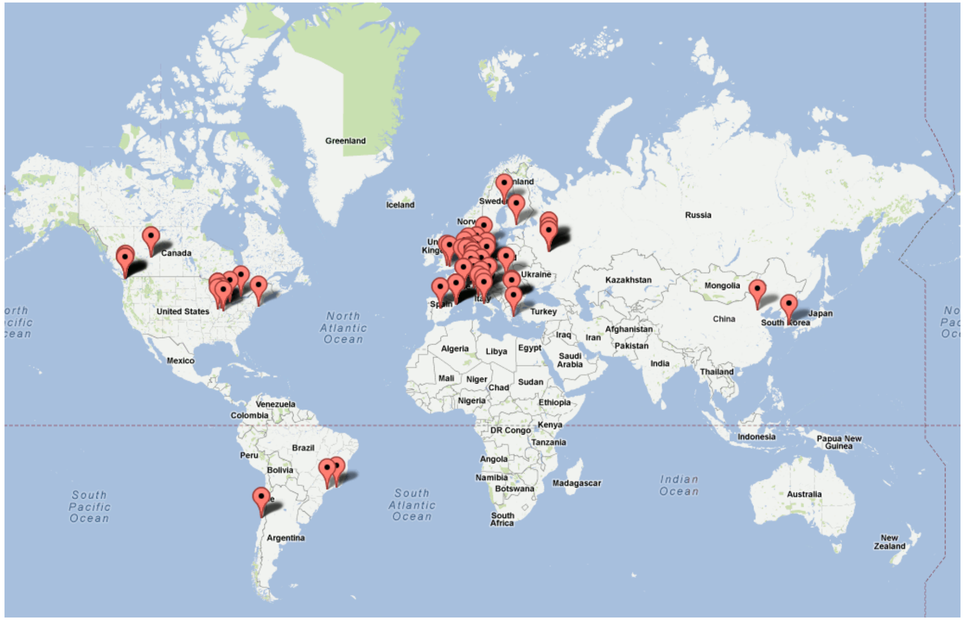
In partnership with:

dCache Mission Statement

"... to provide a system for storing and retrieving huge amounts of data, distributed among a large number of heterogeneous server nodes, under a single virtual filesystem tree with a variety of standard access methods."

https://www.dcache.org/about

dCache Usage Worldwide



- 76 installations
- accounts for more than 50% of disk space available for LHC
- also used by other sciences, including:
  - astrophysics
  - neutrino physics
  - photon science
  - genomic research

This broad spectrum of users means accommodating different workflows, access, capacity and data preservation requirements, resources and budgets.

"Quality of Service" and dCache

There can be a considerable difference in the scientists' desires and expectations about how this data is stored and made available for analysis:

- durability (likelihood of data loss);
- total bandwidth (aggregated over all clients);
- bandwidth available to a single client;
- access latency;
- some combination of these factors.

Can be based on:

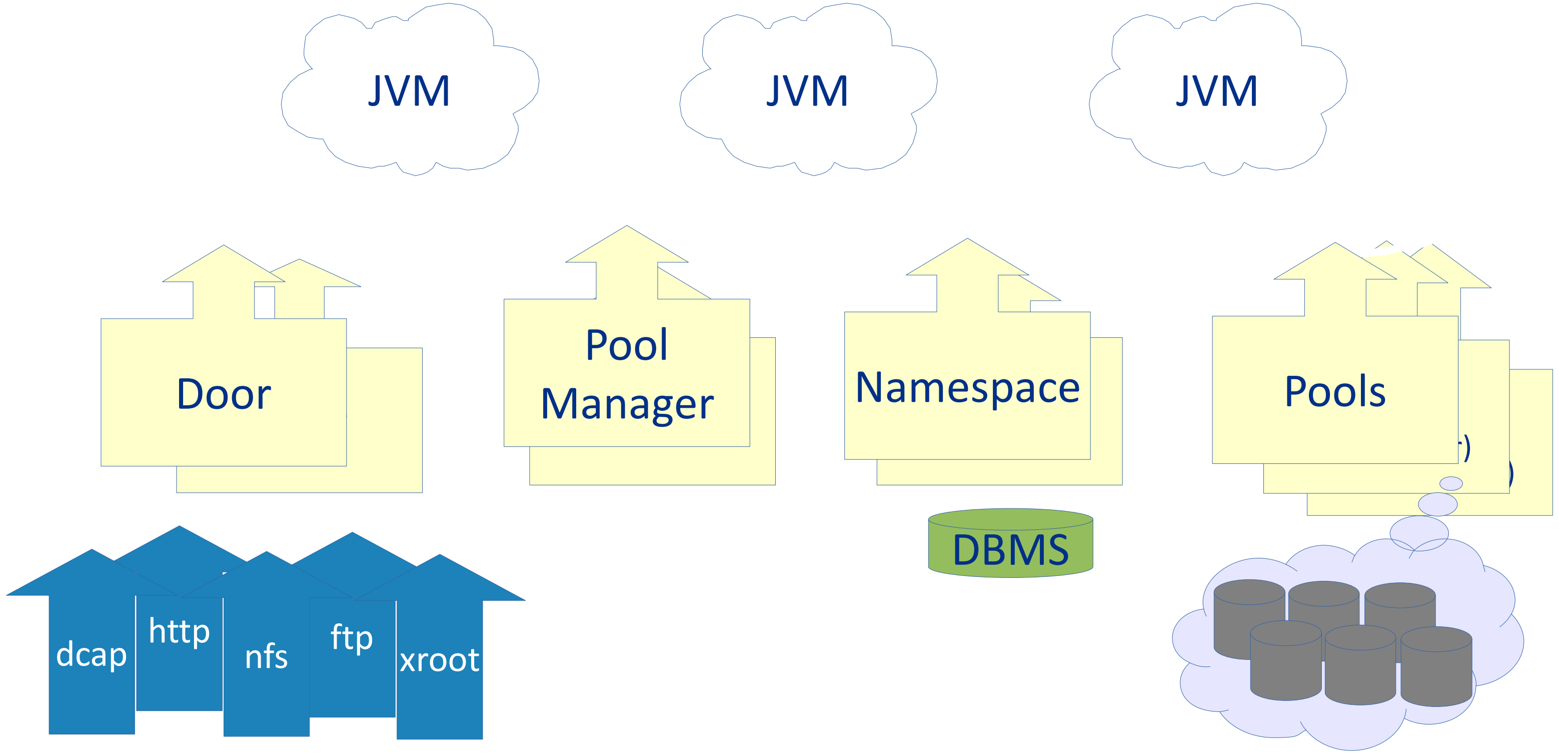
- intrinsic properties of the data;
- types of activity using the data.

"Quality of Service" and dCache

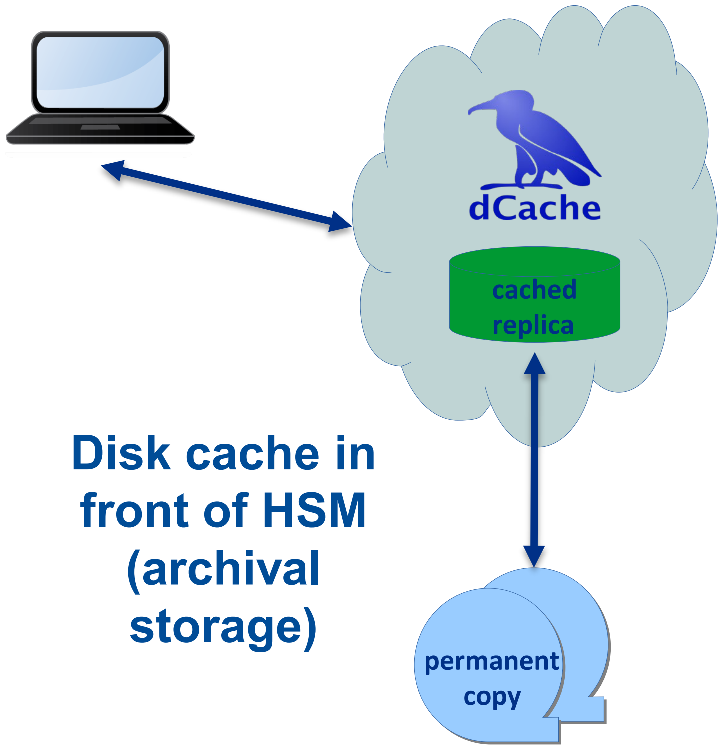
- Not practical or necessary to expose all combinations of options.
- Need to organize these into a common set of agreed-upon QoS classes.
- By allowing scientists to choose and modify with which QoS class they would like their data stored, a storage system gives scientists the ability to achieve the optimal storage strategy within the available storage capacity.

The following is part of an ongoing effort to enhance dCache's capacity to respond to the QoS needs of the communities which deploy it.

dCache in one slide

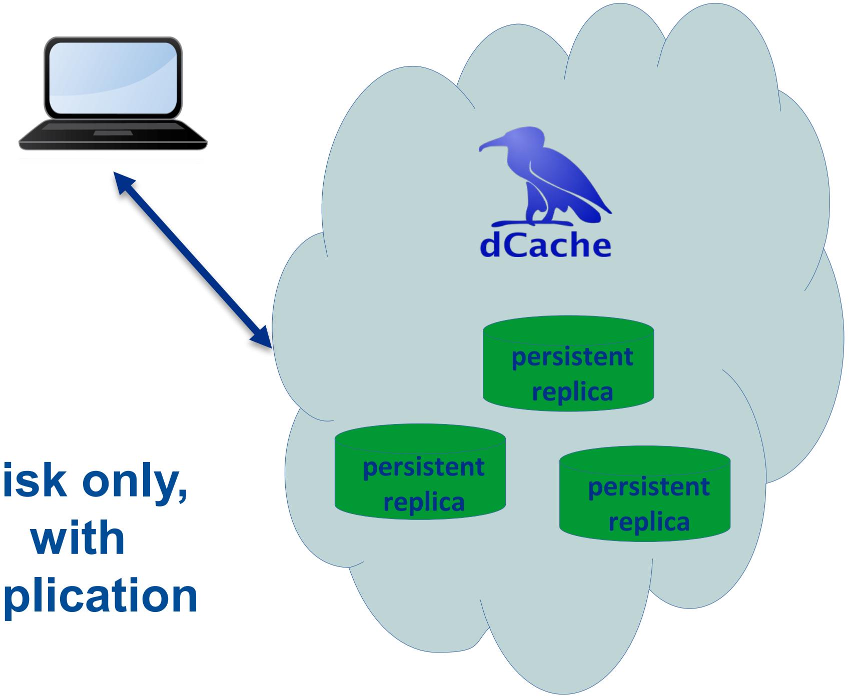


dCache Usage: Two Common Scenarios



Disk cache in front of HSM (archival storage)

Disk only, with replication



dCache currently provides ability to:

1. store data to tape and cache a copy on disk;
2. replicate cached (i.e., temporary) copies of the data if it is being accessed heavily;
3. "pin" data on disk for a determinate period on behalf of user(s);
4. restore data from tape to disk;
5. generate multiple *persistent* replicas of data on disk;
6. create new *persistent* replicas of data if a pool containing a replica goes offline (and remove them when that pool comes back online); this can also involve staging back the data if all such replicas are unavailable and the data is also on tape.

We refer to (5) and (6) as "data resilience".

dCache "Resilience"

- dCache subsystem that achieves data durability by maintaining permanent disk replicas independently of the presence of a back-end or tertiary storage system;
- relies on a partitioning system into resilient and non-resilient pool groups, with file replication managed only within the former;
- seen from the QoS perspective, this looks like the (static) handling of a subset of QoS classes (in other words, keeping files in certain storage groups on disk for faster access).

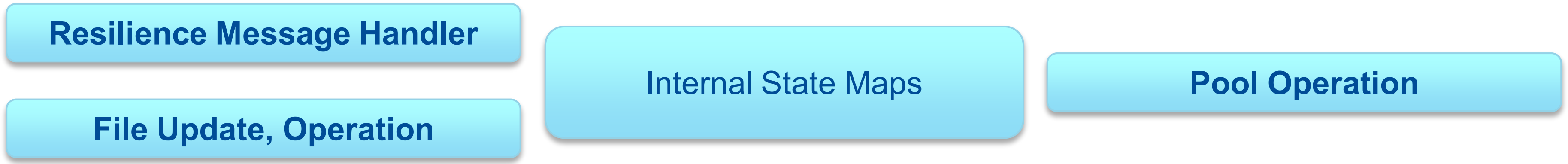
Resilience to QoS

The design of Resilience is potentially extensible to other QoS transitions, but the current implementation creates obstacles to this.

<u>Objectives</u>

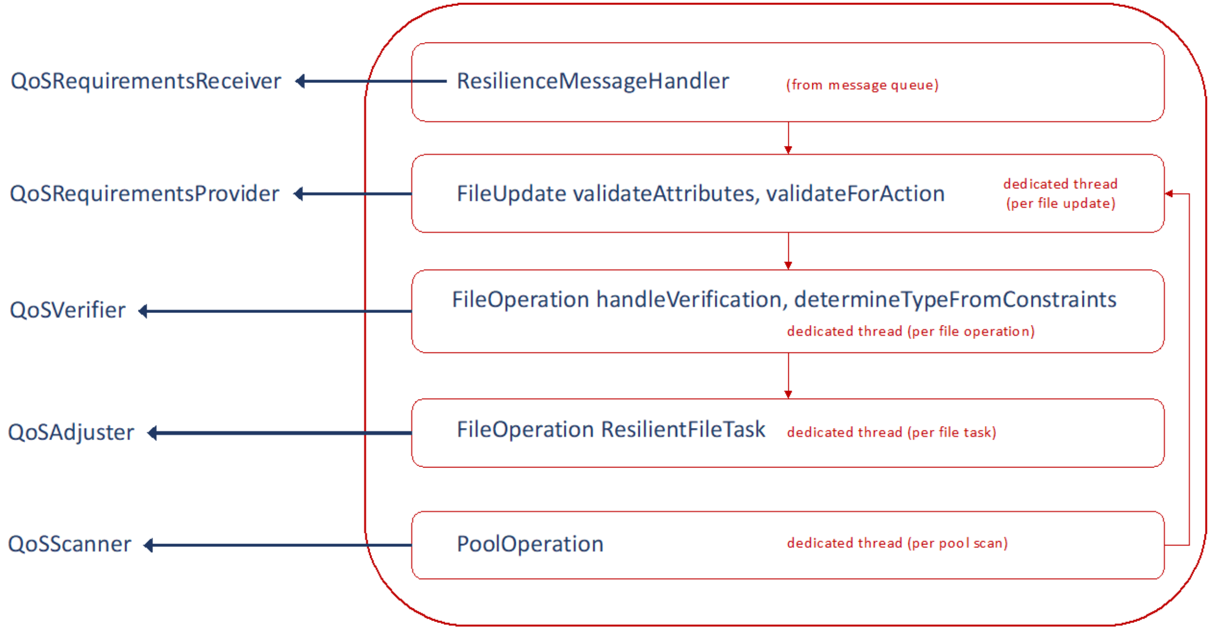
1. Transform Resilience into a set of QoS components, retaining all previous Resilience functionality, but no longer requiring the segregation of "resilient" from "non-resilient" data.
2. Effect a clear separation of layers enabling an API capable of extension without alteration of underlying infrastructure.
3. Prepare the way for a full-fledged "QoS Engine" in which files can be transitioned between states based on a set of time-bound rules.

Resilience Architecture



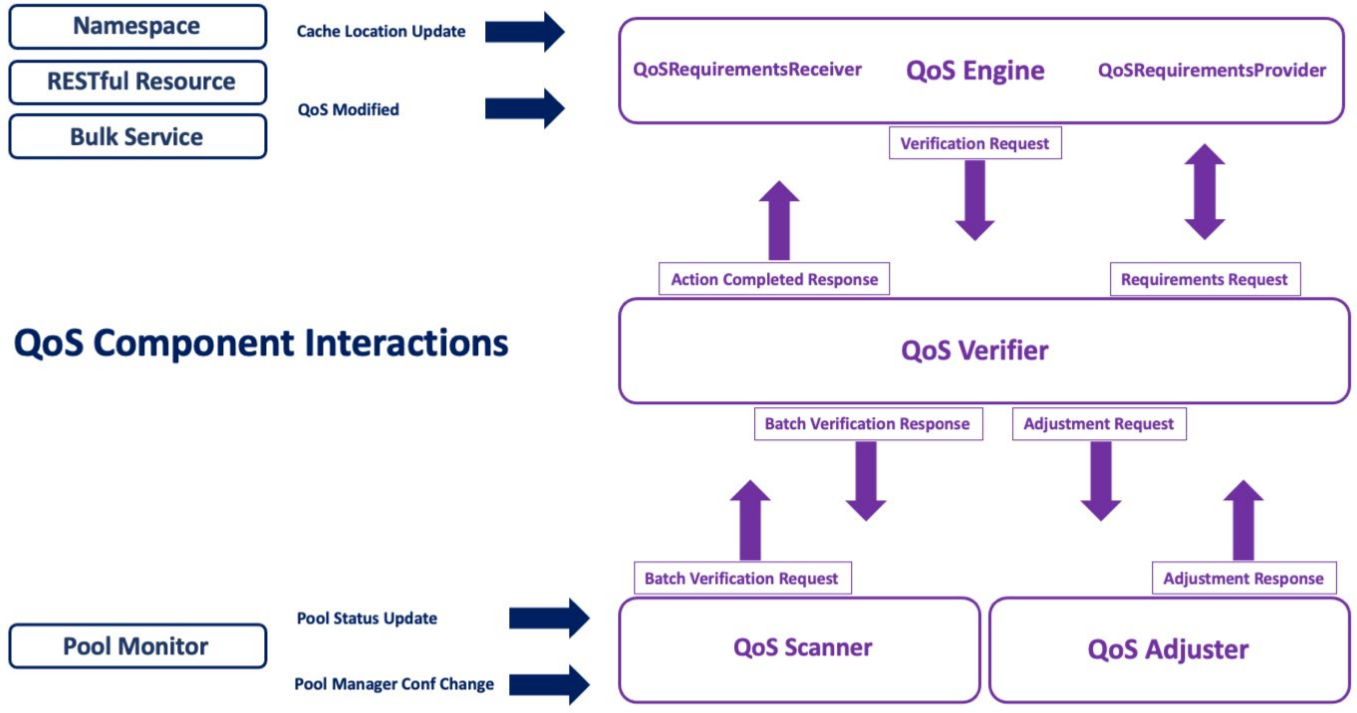
- Responsible for maintaining the required number of replicas on disk, even when pools go offline or come back online.
- Tight integration dictated by this limited purpose, by maximization of efficiency.
- Later discovered issues requiring us to break this tight integration.
- Those modifications now permit complete separation into independent components.

QoS Equivalents


QoSRequirementsReceiver
ResilienceMessageHandler
(from message queue)
QoSRequirementsProvider
FileUpdate validateAttributes, validateForAction
dedicated thread (per file update)
QoSVerifier
FileOperation handleVerification, determineTypeFromConstraints
dedicated thread (per file operation)
QoSAdjuster
FileOperation ResilientFileTask
dedicated thread (per file task)
QoSScanner
PoolOperation
dedicated thread (per pool scan)

1. **Receiver**: receives messages concerning new files and file QoS changes.
2. **Provider**: queried by file and returns the file's requirements.
3. **Verifier**: checks the current status of a file and recommends actions, if any.
4. **Adjuster**: receives an actionable request for a single operation/transformation (STAGE, FLUSH, COPY, CACHE).
5. **Scanner**: receives messages concerning pool status changes and schedules periodic scanning of pools.

QoS Component Interactions



- **Receiver** and **Scanner** directly converted; other components required pulling apart tightly coupled interactions.
- Can be run as separate services or as single standalone service.
- **Verifier** is the heart as in Resilience; **Engine** is the entry point.

Separation into Components

Allows easier redefinition of the QoS Engine "peripherals":

1. **Adjuster**: simple tasks which rely on other parts of dCache to do the heavy lifting; clear separation of concerns will be crucial when optimized restore scheduling is in place.

2. **Provider**: a major motivation for this refactoring; allows for integration with a separate "rule engine".

Rule Engine Prototype

Uses the current combination of namespace attributes (**Access Latency** and **Retention Policy**) plus membership in a **storage group**, which expresses the number and distribution of replicas, to define QoS classes.

Prototype Mapping of dCache Attributes to QoS Classes

ACCESS LATENCY	RETENTION POLICY	storage unit -required	storage unit -onlyOneCopyPer	QOS	Description
NEARLINE	REPLICA	N/A	N/A	volatile	could be removed at any time
NEARLINE	CUSTODIAL	N/A	N/A	tape	on tape; disk copy could be removed at any time
ONLINE	REPLICA	undefined, 1	N/A	disk	persistent on disk but not written to tape
ONLINE	REPLICA	k > 1	partitioned by tags	disk	k replicas persistent on disk but not written to tape
ONLINE	CUSTODIAL	undefined, 1	N/A	disk+tape	persistent on disk and one copy on tape
ONLINE	CUSTODIAL	k > 1	partitioned by tags	disk+tape	k replicas persistent on disk and one copy on tape

Rule Engine Prototype

On this basis, the following transitions are made available.

Currently can be achieved for single files and for bulk operations, through **RESTful API**.

Prototype QoS Transitions and How They are Implemented

QOS TRANSITION	CHANGE IN NAMESPACE	WHAT HAPPENS
volatile => disk	NEARLINE REPLICA => ONLINE REPLICA	k replicas are copied or made “sticky”
volatile => tape	NEARLINE REPLICA => NEARLINE CUSTODIAL	file is migrated to tape-backed pool, if necessary, and then flushed
volatile=>disk+tape	NEARLINE REPLICA => ONLINE CUSTODIAL	file is migrated to tape-backed pool, if necessary, and then flushed; k replicas are copied or made “sticky”
disk => tape	ONLINE REPLICA => NEARLINE CUSTODIAL	file is migrated to tape-backed pool, if necessary, and then flushed; all replicas are cached
disk => disk+tape	ONLINE REPLICA => ONLINE CUSTODIAL	file is migrated to tape-backed pool, if necessary, and then flushed
tape => disk	NEARLINE CUSTODIAL => ONLINE REPLICA	NOT SUPPORTED
tape => disk+tape	NEARLINE CUSTODIAL => ONLINE CUSTODIAL	LOCALITY = ONLINE_NEARLINE (file is on disk): k replicas are made sticky or copied if not enough cached replicas already exist
tape => disk+tape	NEARLINE CUSTODIAL => ONLINE CUSTODIAL	LOCALITY = NEARLINE (file not currently on disk): file is staged from tape; k replicas are copied
disk+tape => tape	ONLINE CUSTODIAL => NEARLINE CUSTODIAL	all replicas are cached
disk+tape => disk	ONLINE CUSTODIAL => ONLINE REPLICA	NOT SUPPORTED

Rule Engine Prototype Limitations

1. Only the namespace attributes can be changed dynamically for single files (via a query); the number of copies is statically defined by storage unit.
2. No provision for indicating the number or distribution of copies that should reside on tertiary storage.
3. No component which could be given time-based rules concerning how and when an individual file's QoS should be changed.

Overcoming the coarseness of these semantics will be a major goal for future dCache QoS development.

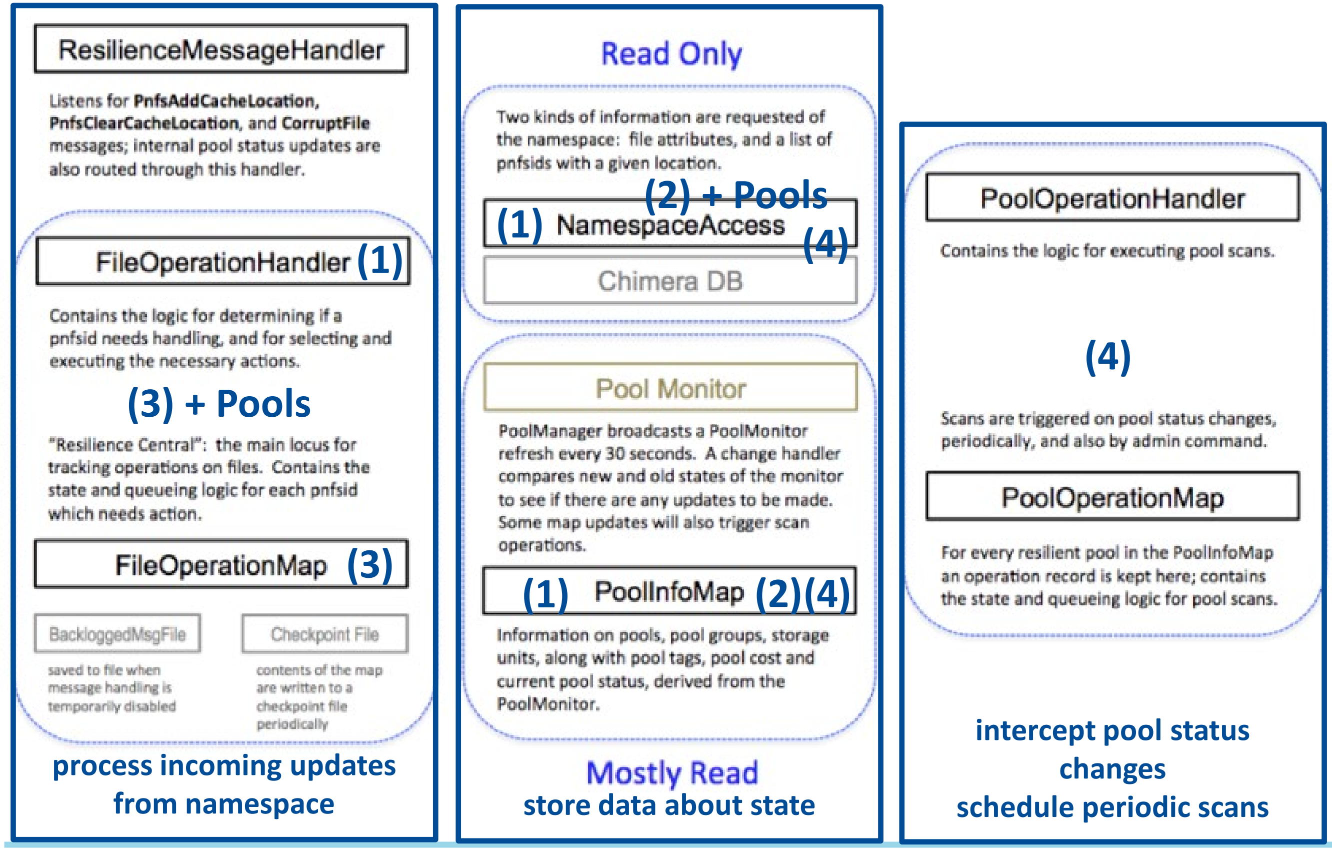
Acknowledgments

This work is supported by the Fermi National Accelerator Laboratory, managed and operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes.

Thank you for listening. Questions?

Fermilab

Resilience Component Responsibilities



ResilienceMessageHandler

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

FileOperationHandler (1)

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

(3) + Pools

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

FileOperationMap (3)

BackloggedMsgFile: saved to file when message handling is temporarily disabled

Checkpoint File: contents of the map are written to a checkpoint file periodically

process incoming updates from namespace

Read Only

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

(2) + Pools

(1) NamespaceAccess (4)

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.

(1) PoolInfoMap (2)(4)

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

Mostly Read

store data about state

PoolOperationHandler

Contains the logic for executing pool scans.

(4)

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.

intercept pool status changes
schedule periodic scans

1. **how many replicas are required?**

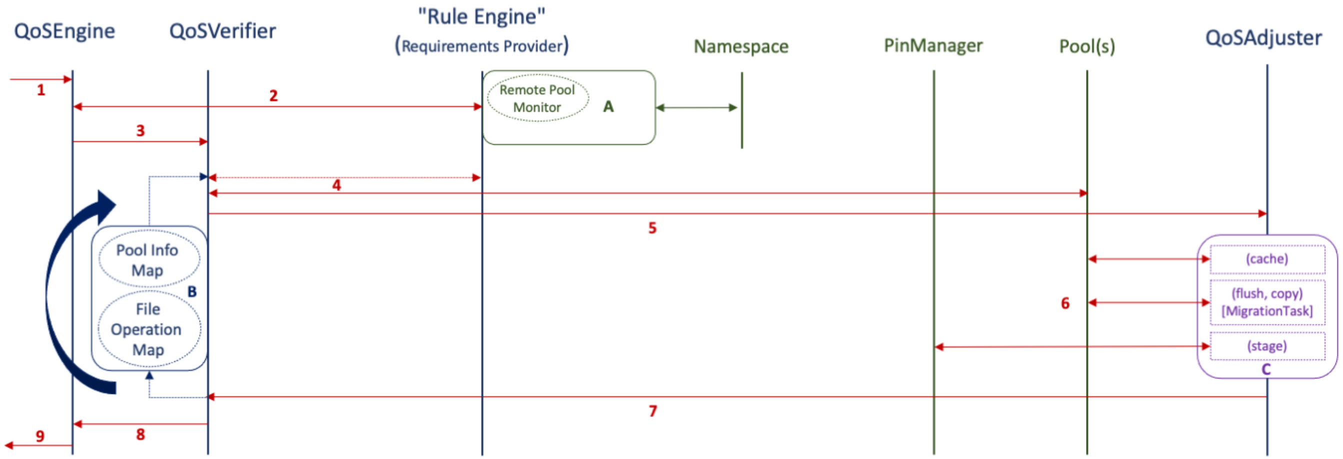
2. **how many replicas are currently accessible?**

3. **make the necessary adjustments**

4. **react to a change in pool state or periodically verify for consistency**

DESY. neic dCache

QoS Request Handling (Messaging)



1. Receive message (cache update or QoS modification).
2. Check the file requirements.
3. Request verification.
4. Verify the status of the file (how many replicas, on tape, etc.) on the pools (and recontact provider on iteration).
5. Determine action and possibly request adjustment.
6. Process the task; if it fails, possibly retry.
7. Notify verifier of success/failure; verifier reevaluates for further action (retry, continue to new action, quit).
8. Remove operation and send verification/action completed message.
9. Notify QoS transition completed (topic).

A - the current rule engine uses the namespace and Pool Selection Unit. B - The pool selection is done by the verifier and sent as part of the message to the adjuster; verifier keeps track of maximum running slots and only sends ready tasks to the adjuster. C - The adjuster queues the requests, maps them to adjuster types and executes; the types call out to either the pools or the PinManager.