From Resilience to QoS

Albert L. Rossi (FNAL)

From Resilience to QoS

The Resilience subsystem achieves data durability by maintaining permanent disk replicas independently of tertiary storage.

dCache installation must be partitioned into resilient/non-resilient pool groups.

Seen from QoS perspective, this also looks like (static) maintenance of faster access for a subset of files.

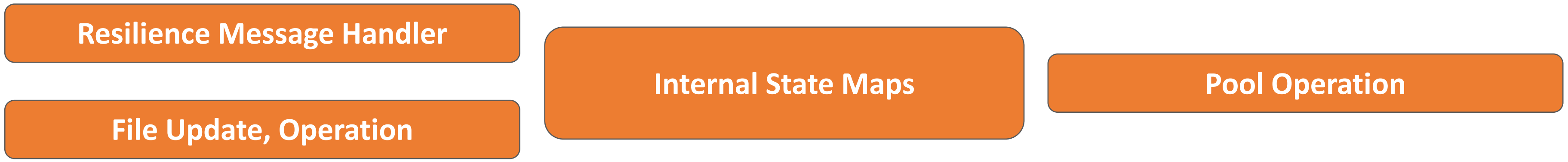
From Resilience to QoS

The Resilience service is potentially extensible to other QoS transitions, but currently its implementation creates obstacles.

Working goals:

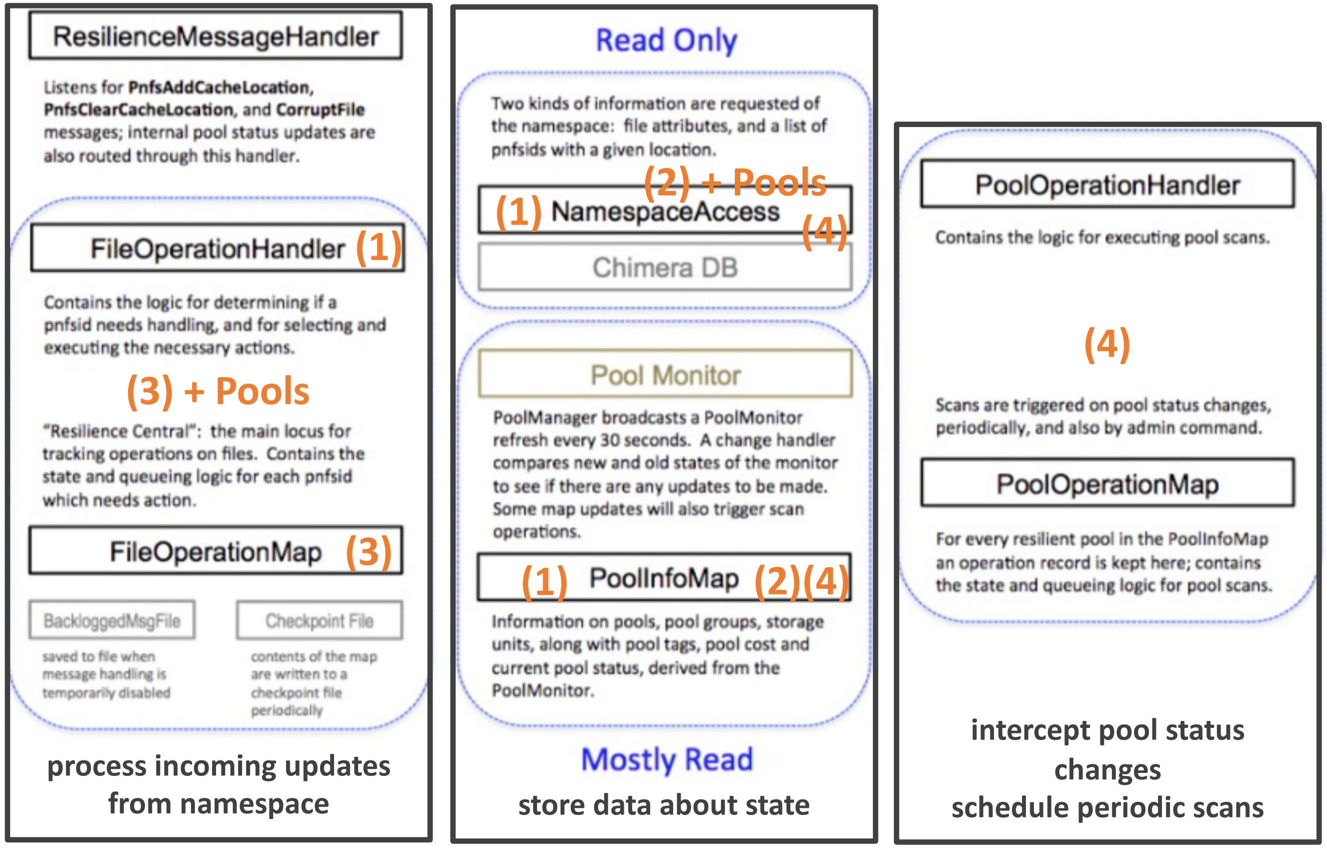
- Transform into a set of QoS components, retaining Resilience functionality but no longer requiring data segregation into resilient vs non-resilient.
- Separate layers in a way allowing for extension, preparing for a full-fledged "Engine" in which files can be transitioned between QoS classes/states based on rules.

Resilience Component Responsibilities



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

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

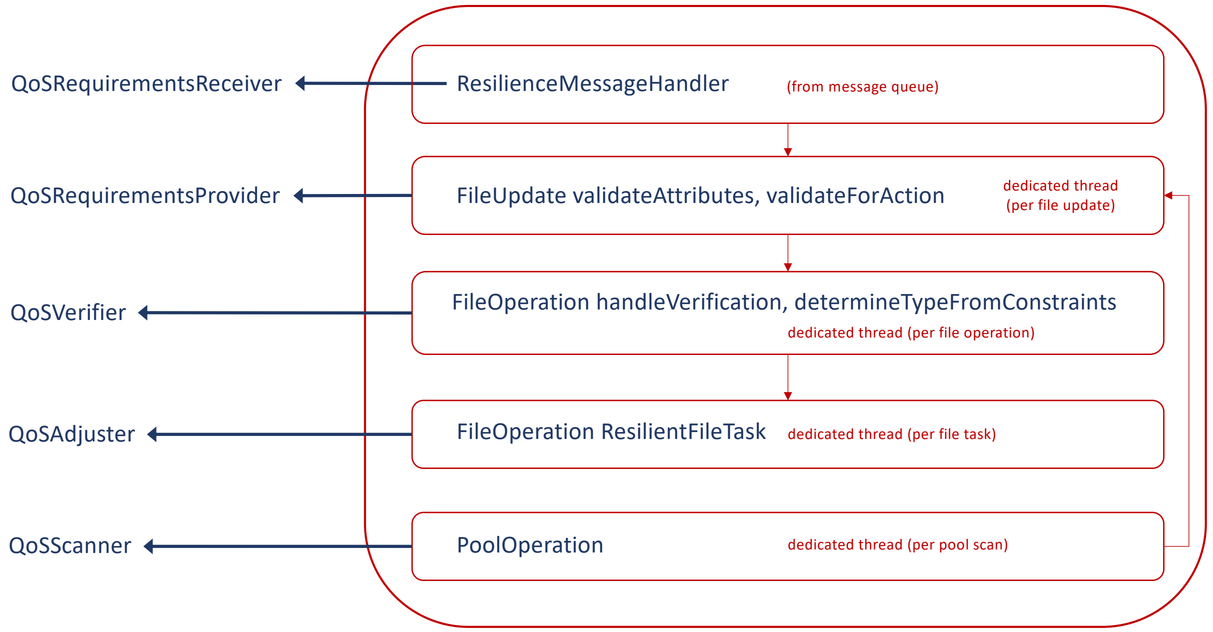
1. how many replicas are required?

2. how many replicas are currently accessible?

3. make the necessary adjustments

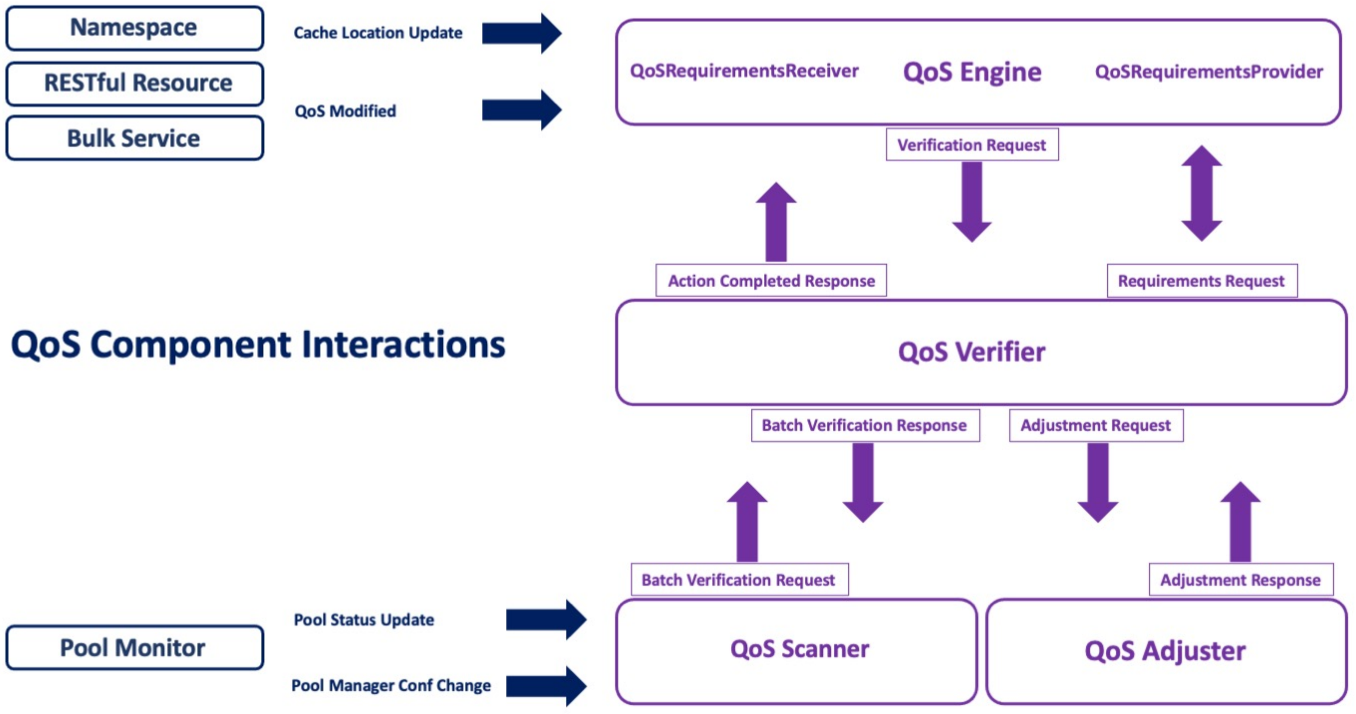
4. react to a change in pool state or periodically verify for consistency

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)

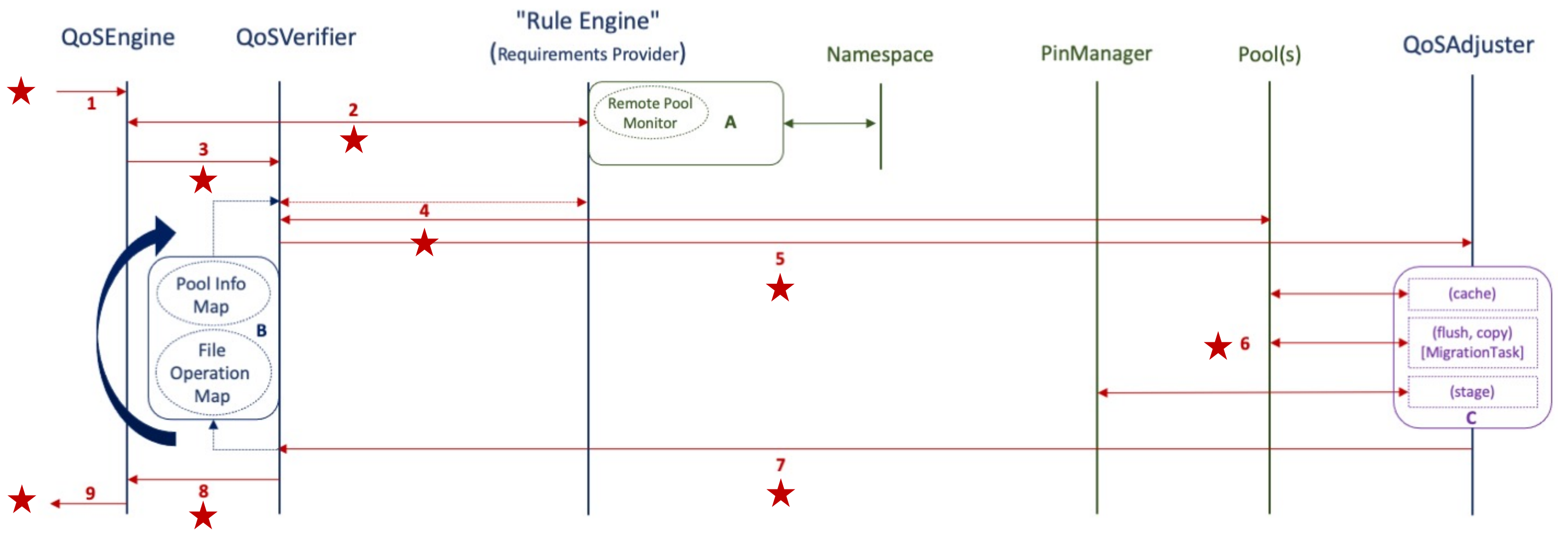
1. **Receiver**: receives messages concerning new files and file QoS changes.
2. **Provider**: queried by file and returns the file's requirements.
3. **Verifier**: checks the current status of a file and recommends actions, if any.
4. **Adjuster**: receives an actionable request for a single operation/transformation (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.

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.

Separation into Components

Allows easier redefinition/modification of the QoS Engine "peripherals":

- **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 (Lea's work/presentation).
- **Provider**: a major motivation for this refactoring; allows for integration with a separate "rule engine".

Rule Engine Prototype

Uses the current combination (from Resilience) of namespace attributes (**Access Latency** and **Retention Policy**) plus membership in a storage group (**storage unit**) expressing the number and distribution of disk replicas, to define a set of very basic QoS classes.

Prototype Mapping of Attributes to 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.

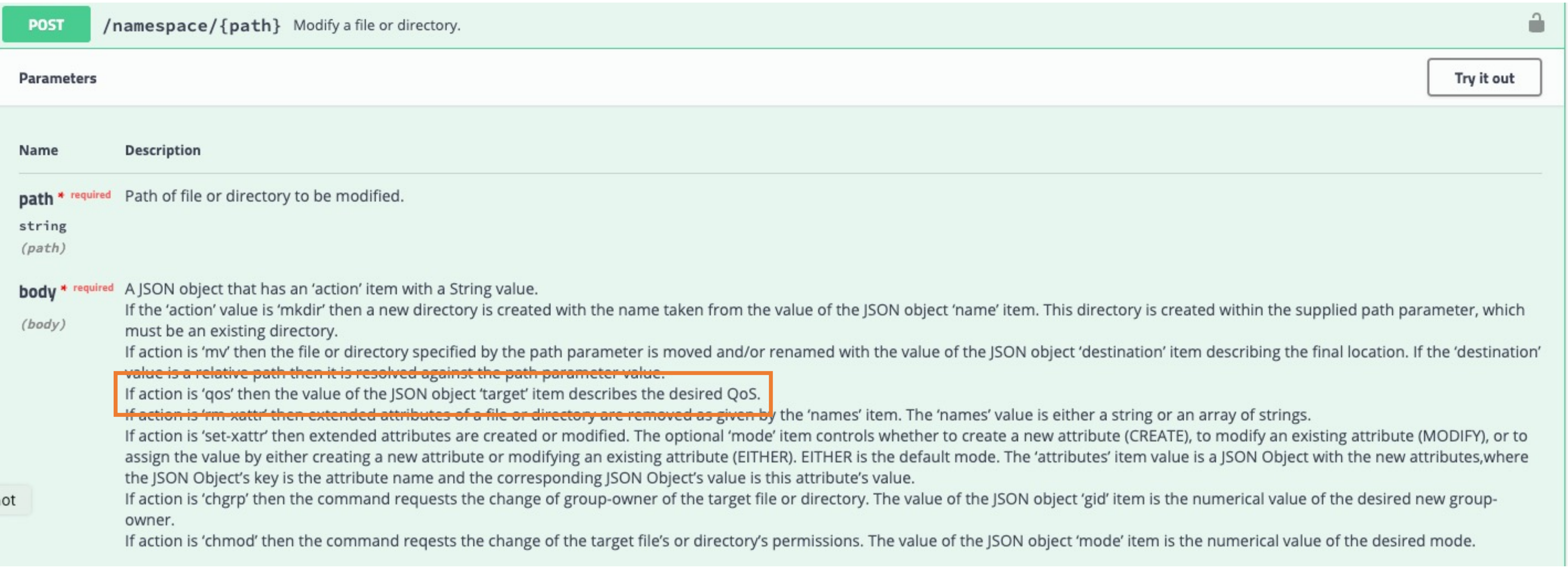
These can be requested for single files and for bulk sets through a RESTful API (dCache **Frontend** service).

Prototype Transitions and Implementation

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

"bring online"

Rule Engine Prototype



POST /namespace/{path} Modify a file or directory.

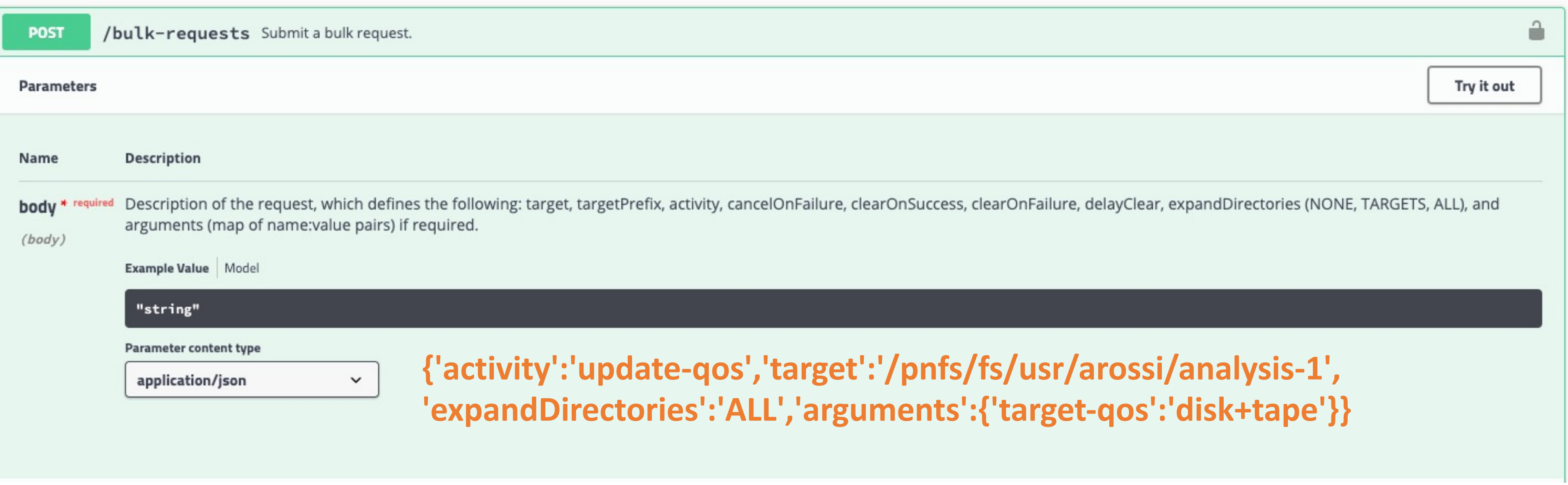
Parameters

Try it out

Name	Description
path * required string *(path)*	Path of file or directory to be modified.
body * required *(body)*	A JSON object that has an 'action' item with a String value. If the 'action' value is 'mkdir' then a new directory is created with the name taken from the value of the JSON object 'name' item. This directory is created within the supplied path parameter, which must be an existing directory. If action is 'mv' then the file or directory specified by the path parameter is moved and/or renamed with the value of the JSON object 'destination' item describing the final location. If the 'destination' value is a relative path then it is resolved against the path parameter value. If action is 'qos' then the value of the JSON object 'target' item describes the desired QoS. If action is 'rm-xattr' then extended attributes of a file or directory are removed as given by the 'names' item. The 'names' value is either a string or an array of strings. If action is 'set-xattr' then extended attributes are created or modified. The optional 'mode' item controls whether to create a new attribute (CREATE), to modify an existing attribute (MODIFY), or to assign the value by either creating a new attribute or modifying an existing attribute (EITHER). EITHER is the default mode. The 'attributes' item value is a JSON Object with the new attributes,where the JSON Object's key is the attribute name and the corresponding JSON Object's value is this attribute's value. If action is 'chgrp' then the command requests the change of group-owner of the target file or directory. The value of the JSON object 'gid' item is the numerical value of the desired new group-owner. If action is 'chmod' then the command reqests the change of the target file's or directory's permissions. The value of the JSON object 'mode' item is the numerical value of the desired mode.

(from our Swagger page: https://host:3880/api/v1)

Rule Engine Prototype



/bulk-requests Submit a bulk request.

Parameters

Name	Description
body * required (body)	Description of the request, which defines the following: target, targetPrefix, activity, cancelOnFailure, clearOnSuccess, clearOnFailure, delayClear, expandDirectories (NONE, TARGETS, ALL), and arguments (map of name:value pairs) if required.

```
"string"
```

Parameter content type

application/json

```
{'activity':'update-qos','target':'/pnfs/fs/usr/arossi/analysis-1',
'expandDirectories':'ALL','arguments':{'target-qos':'disk+tape'}}
```

(from our Swagger page: https://host:3880/api/v1)

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.