REST API for bulk operations

This document describes a proposed extension to dCache's existing REST API (frontend) to support bulk operations. A bulk operation is a request that targets multiple resources.

When a client uses a bulk operation, the result is called a bulk request.

A bulk request is handled asynchronously. For details, see the Processing Model section.

The kind of bulk operation is called the activity. Pinning is one example of an activity, deleting content is another example.

Goals

1. It should be easy to add extra operations in the future without modifying existing support for bulk operations.
2. It should be possible to target a directory (and everything within it).
3. It should be possible to target an arbitrary list of files.
4. A client should be able to make other requests while a bulk request is being processed. This should be possible without establishing new TCP connections.
5. A client should be able to check the progress of the bulk request.
6. A client should be able to cancel a bulk request and observe how far the request proceeded.
7. There is a maximum number of concurrent bulk operations. This maximum value is enforced per user (unique uid).
8. A user should be able to discover all ongoing and completed bulk requests.

Non-goals

It is not possible to make a bulk request with multiple activities. To achieve multiple activities, a client must make multiple bulk requests.

There is no mechanism for a client to receive (asynchronous) notification from the progress of bulk requests. Such a feature may be added in the future using (for example) SSE.

Generic API framework

This section describes the different resources with which a client may interact, which request types are supported, and how those requests are processed.

Resource /api/v1/`bulk`

This resource represents all bulk operations.

GET requests

Request entity

No entity in the request

Query String

The request URL may include a query string. If present the query string value has the form key=value pairs with an ampersand separating each pair. This has the effect of limiting or filtering the response.

The following keys are accepted in the query string:

Name	Value	Result
`status`	A comma-separated list of non-repeating elements, each of which is one of: `queued`, `started`, `completed`, `cancelled`.	Returns only those bulk requests that have one of the supplied status.

Response

Response is a 401 (Unauthorized) status code if the client is anonymous.

Otherwise, the response has MIME-Type `application/json` and the response entity is a JSON array of JSON Strings. Each JSON String is an absolute URL of a bulk request made by this user and that have not been cleared.

If the client included no query string then the response contains all bulk requests made by this user that have not been cleared. If the user has made no bulk requests or all bulk requests have been cleared then the response is an empty array.

If the client specified a query string then the response contains all bulk requests that match the query string arguments and have not been cleared. If the user has no bulk requests that match the query string and have not been cleared then the response is an empty array.

POST requests

Request entity

Required entity in the request, must have MIME-Type `application/json` and must be a JSON object. The following elements are supported:

Name	Format	Required?	Purpose
`target`	JSON String or Array of	Yes	If the value is a JSON Array of JSON Strings then each array element is the relative or

	Strings		absolute path of a file or directory within dCache that this bulk request will affect. If a path is relative then `target_prefix` is used to resolve this path to an absolute path. If a bulk request has a single target then the `target` value may be a JSON String containing the target's path.
`target_prefix`	JSON String	No	An absolute path used to resolve relative paths in `target`. Default: /
`activity`	JSON String	Yes	How the targets will be updated.
`clear_on_success`	JSON Boolean	No	Whether a bulk request is automatically cleared on successful completion. Default: false
`clear_on_failure`	JSON Boolean	No	Whether a bulk request is automatically cleared on partial successful or unsuccessful completion. Default: false.
`delay_clear`	JSON Integer	No	The number of seconds to delay automatic clearing of a bulk request. At least one of `clear_on_success` and `clear_on_failure` must be true. Default: 0.
`arguments`	JSON Object	Depends on `activity` value.	Qualifies how targets are updated. Whether this is required and the precise format of the corresponding JSON Object depends on the `activity` value. Default: an empty JSON Object.
`recursive`	JSON Boolean	No	If true then a directory listing is made for each directory target. The contents of that directory target are added to the existing target list without creating duplicates. Any added directory targets are also expanded. This continues until no further directory targets are added. The bulk request will affect all targets. If false then no additional targets are added. Default: false.

Response

If the user is unauthenticated then the server will respond with a 401 (Unauthorized) status code.

If the user is authenticated, but is not authorised to make bulk operations then the server will respond with a 403 (Forbidden) status code.

If the user is authenticated and authorised, but has too many uncleared bulk operations then the server will respond with a 429 (Too Many Requests) status code.

If the user is authenticated, authorised, does not have too many uncleared bulk operations but the supplied JSON Object entity is badly formatted then the server will respond with a 400 (Bad Request) status code.

If the user is authenticated, authorised, and has space capacity to make bulk requests and the JSON object is correctly formatted then the server will accept the bulk request and respond with a 201 (Created) status code. This response will include a Location HTTP response header with a value that is the absolution URL for the resource associated with this bulk request.

Resource /api/v1/`bulk/<id>`

This resource represents a specific bulk request, where `<id>` is some unique identifier for some corresponding bulk request.

The server will respond to all requests where the client has failed to authenticate with a 401 (Unauthorized) status code.

The server will respond to all requests where `<id>` has no corresponding bulk request, or the corresponding bulk request has been cleared, or is not owned by the authenticated user with a 404 (Not Found) status code.

GET requests

Request entity

No entity in the request

Response

The response has MIME-Type `application/json` and the response is a JSON Object that describes the current status of the bulk request.

The following elements may be present:

Name	**Format**	**Included?**	**Description**
`status`	JSON String	Yes	One of `queued`, `started`, `completed`, `cancelled`. ● `Queued` indicates that the request has been accepted but no work has been made yet. ● `Started` indicates that work on the request is underway. ● `Completed` indicates that all targets were attempted and no further work on the request will take place.

			• `Cancelled` indicates that the request was stopped before reaching completed.
`targets`	JSON Integer	Yes for recursive requests, Never otherwise.	The number of targets discovered so far.
`processed`	JSON Integer	Yes for status `started`, `completed` and `cancelled`, otherwise never.	The number of targets where the requested activity has either been started or has completed (successfully or otherwise). This number is always greater than or equal to the number of targets listed in `failures`.
`failures`	JSON object	Yes if at least one target has failed, otherwise never.	The failed targets for this bulk request.
`request`	JSON object	Yes	The client-supplied JSON Object that triggered this bulk operation.

Describing failures

The failures JSON Object, if present, has elements with names that describe a failure mode that at least one target suffered. The corresponding value is a JSON Array of JSON Strings that list the targets that suffered that failure mode.

If the bulk request contains a target prefix (the `target_prefix` element in the request) and the failed target is located within that prefix then the JSON String value is a relative path: the target path relativised using the supplied prefix.

PATCH requests

Request entity

The entity must have MIME-Type `application/json` and the entity must be a JSON Object with element `action` that has a JSON String as a value.

If the action is `cancel` then no further elements are needed in the JSON Object.

Processing

If the action is `cancel` then:

- If the bulk operation is in state `started` then all dCache activity for this bulk request is stopped.

- The corresponding bulk request status is updated to `cancelled` if it is currently `queued` or `started`. It does not change in the status is `cancelled` or `completed`.

Response

If the request is badly formed or the action value is unknown then the server will respond with a 400 (Bad Request) status code.

Otherwise the response has a status code of 200 (OK).

DELETE requests

Request entity

No entity in the request.

Processing

If the bulk operation was in state `started` then all dCache activity triggered by this bulk request is stopped.

The bulk request is cleared. No further activity will take place for this request. The server will respond to subsequent GET requests targeting this resource with a 404 (Not Found) status code.

Response

The server responds with a status code 204 (No Content).

Processing model

A client creates a bulk request by making a POST request to the `/api/v1/bulk` resource. If the request is well-formed and the user is both authorised to make bulk requests and has not exceeded the number of concurrent bulk requests then the request is accepted.

A bulk request could target multiple files or directories that have a common prefix; for example, several files within the same directory. In this case, this common prefix may be specified in the bulk request, allowing the list of targets to use relative paths, where appropriate. This will reduce the size of the JSON Object.

The interface supports bulk requests being queued; that is, after accepting a bulk request, dCache may decide to delay working on the request. This feature is optional and may be omitting in the initial version.

The interface also supports recursive requests, where directory targets are expanded to include their child elements. This expansion may be done first, to build a complete list of targets. Alternatively, in some cases, the expansion may be done in parallel with the activity to speed up operation and reduce memory overhead.

It is possible that the activity fails for some or all of the targets. These are reported back in the status, via the response to a GET request. In the case of recursive requests, this response may be large.

The interface includes the concept of clearing a bulk request. This is achieved by making a DELETE request to the bulk request's resource. Clearing is the process by which all resources associated with processing the request are freed. This allows the client to discover whether a request was successful and, if not, which targets failed.

The API also supports automatic clearing of bulk requests. The main reason to include this is to avoid that bulk requests hang around indefinitely, should the client loose track of them. It is anticipated that a client would request automatic clearing after a relatively large time (e.g., one hour).

Bulk activity

This section describes the various types of activity that the client may request.

Pin

The `pin` activity is used to pin one or more files. It has no effect on directories.

The `arguments` element in the request is optional. If present and contains the `lifetime` element then the value is a JSON Number describing the number of seconds the file should be pinned. The value must be greater than or equal to `0`.

A missing `lifetime` element or `arguments` element is equivalent to specifying a value of `0`.

A `lifetime` value of `0` will stage the file, but not guarantee the file is available immediately after the stage is complete.

Unpin

The `unpin` activity is used to remove all pins owned by the current user on one or more files. It has no effect on directories.

Delete

The `delete` activity is used to delete a file or directory. Attempting to delete a non-empty directory will fail. Recursive requests are processed depth-first; i.e., expanded child elements are processed before the parent target.

Delete-children

The `delete-children` activity is used to delete the contents of directory targets. It has no effect on file targets. Attempting to delete a non-empty directory will fail. Recursive requests are processed depth-first; i.e., expanded child elements are processed before the parent target.

Archive

The archive activity is used to create a single file that contains the file targets. Depending on the archive format, directory targets may be included.

The request requires an `arguments` element. The following table describes the format of the `arguments` element.

Name	**Format**	**Required?**	**Description**
`format`	JSON String	Yes	One of **zip**, **tar**, **tgz**. This describes what type of archive is created. Additional acceptable values may be added in the future.
`path`	JSON String	Yes	An absolute path within dCache for the archive file. The parent directory must already exist. If the path lies within a recursive target then the archive namespace entry is not included in the archive content.
`root`	JSON String	No	This value describes a directory that all archived contents will be referenced against. The value must be a directory and must be an ancestor of all targets Default: /. For example, consider an archive is created with the file `/experiments/foo/data/2019/bar/baz.h5`. Without specifying a `root` value, this file is added with its absolute path. If `root` is specified as `/experiments/foo/data/2019` then the file is added as `/bar/baz.h5`.

TODO format-specific options? E.g., compression level in zip.

Question: support creating an archive that contains only PNFS-IDs of files?

Question: support arbitrary file placement within the archive?

Update-qos

The `update-qos` activity updates the QoS of file targets. It does not affect directory targets.

The activity requires the request JSON to have an `arguments` element, with one mandatory element: `target-qos`. The `target-qos` element has a JSON String as an argument, which describes the desired QoS for file targets.

Default-qos

The `default-qos` activity updates the default QoS for files written in directory targets. It does not affect file targets.

The activity requires the request JSON to have an `arguments` element, with one mandatory element: `target-qos`. The `target-qos` element has a JSON String as an argument, which describes the desired default QoS for files written into directory targets.

Chgrp

The `chgrp` activity updates the group-ownership of targets owned by the user in a manner similar to the chgrp(1) command in Unix/Linux.

The activity requires the request JSON to have an `arguments` element, with exactly one element from two possible elements `group` and `gid`. If supplied, the `group` element has a JSON String as an argument, which describes the new group-owner group name. Otherwise the `gid` element has a JSON Number as an argument describing the new group-owner's gid.

Outstanding issues

VG: What would be the atlas data carousel use case here ? i.e. Move a file from tape to disk (from nearline to online pinned replica on datadisk)

VG: One immediate thought: Then the experiment layers should use this interface. Most likely, FTS doing REST call instead of SRM calls. The current workflow with SRM can be improved as well.

PM. Should the GET response to the request URL include the original request JSON Object? The advantage is that it provides complete information: a client doesn't have to remember what it requested. The disadvantage is that all failed targets will be listed twice: once in the `request` JSON Object and again in the `failures` JSON Object.