Table of Contents

Planned functionality of the prestaging script

The script serves as an actual prestaging tool, as a reference implementation and as a testing tool. What follows describes the desired behaviour, not the current behaviour of the script.

Where can I find it?

- Download the following files:
  - prestage.py
  - gfalrequest.py
  - setup.sh
- Do a `source setup.sh`
- Run `./my_prestage.py [OPTIONS]`
  - `-s surls`: `surls` is a comma-separated list of surls, or
  - `-i surlfile`: `surlfile` is a file containing a list of surls (one per line)
  - `-t space`: `space` is a space token (optional)
  - `-l pinlifetime`: `pinlifetime` is a pin lifetime (optional)
  - `-L`: if used, the script does not use *srmLs*

How is it used?

Given a list of SURLs, the script is executed and runs until one of these conditions is fulfilled:

1. all files are staged (the GFAL status is 1 and the locality is ONLINE or ONLINE_AND_NEARLINE)
2. a timeout is reached (to do)
3. the script is interrupted by the user via CTRL-C

What is the input?

Input parameters can be fed via command line or a configuration script (to do). They include:

- a file containing a list of SURLs
- a timeout (internally used to set desiredTotalRequestTime) (to do)
- a space token descriptor (optional)
- a desired pin lifetime

What does it produce?

Every time the script polls the status of the request, it generates three files:

- `file_status.txt`: a text file with one line per file, each line containing these comma-separated fields:

  ```
  time, surl, gfal_status, srm_status, explanation, pinlifetime, estwaittime, locality
  ```

  where `gfal_status` is the GFAL status (-1=error, 0=pending, 1=done) and `srm_status` is the file level status. Empty fields correspond to None or to unsupported quantities (like `estwaittime` and `srm_status` now). `time` is the time spent since the request submission. The user can decide if `file_status.txt` is overwritten each time or if the new status is appended to it. Note that currently GFAL does not return a `pinlifetime` for BringOnline requests. (to do)

- `request_status_<request token>.txt`: a text file summarizing the status of the request with this format:

  ```
  TOKEN=<request token>
  TIME_TOT=<time passed since request submission>
  TIME_STATUSOF=<time spent on the StatusOf call>
  TIME_LS=<time spent on the Ls call>
  ERRORS=<number of files with status = -1>
  PENDING=<number of files with status = 0>
  DONE=<number of files with status = 1>
  NONE=<number of files NONE>
  LOST=<number of files LOST>
  UNAVAIL=<number of files UNAVAILABLE>
  NEARLINE=<number of files NEARLINE>
  ONLINE=<number of files ONLINE or ONLINE_AND_NEARLINE>
  ```

  All the fields can be put in the same line and a new line appended to the same file at each polling.
- `prestage.log`: a log file where to write error messages and debugging information. (to do)

How does it poll?

The request status is polled at regular intervals when *gfal_prestagestatus* returns 0. The backoff time increases exponentially when *gfal_prestagestatus* returns -1.

StatusOf or Ls?

The GFAL status of the files is defined after a *gfal_prestagestatus*, while the locality can be determined only via a *gfal_ls*. By default both functions are used, but it is possible to choose only to use StatusOf. Using both allows to spot any inconsistencies (like the locality being ONLINE and the GFAL status being 0).

Cleaning up

When the script finishes (including the case where it is killed by the user), it aborts the request and optionally releases all the files (to do). The files should be released when the script is used for testing, and not released when it is used for real prestaging.

Optimizations

Ls is done only on the files which had not reached a final state at the previous polling.

Data analysis

The output data defined above is sufficient to perform a detailed data analysis. Some examples:

- compare performances of Ls and StatusOf
- study and compare the time evolution of the number of ONLINE files and of DONE files
- measure the performances of the underlying tape system
- look for stale files (e.g. files which never become ONLINE).

To-do list

In addition to the things marked above, the following must be done:

- add the poll time as an input parameter (now it is fixed at 60 s)
- add a retry loop when the BringOnline request fails
- print to `request_status_<request token>.txt` error messages with the format:

```
TIME_TOT ERROR <error message>
```

For example:

```
67.362 ERROR [SE][BringOnline] httpg://srmcms.pic.es:8443/srm/managerv2: CGSI-gSOAP: Error readin
```

-- AndreaSciaba - 04 Mar 2009

This topic: Sandbox > PreStagingTestsReferenceImplementationArch
Topic revision: r4 - 2009-04-09 - AndreaSciaba