GRID_LRT Documentation

Release 0.5.1

Alexandar Mechev

May 06, 2019

Contents:

This package is built by Alexandar Mechev and the LOFAR e-infra group at Leiden University with the support of SURFsara. The goals of this package is to enable High Throughput processing of LOFAR data on the Dutch GRID infrastructure. We do this by making a set of tools designed to wrap around several different LOFAR processing strategies. These tools are responsible for staging data at the LOFAR Long Term Archives, creating and launching GRID jobs, as well as managing intermediate data on the GRID storage.

CHAPTER 1

Installation

1.1 Via Python Package Index

Install the package (or add it to your `requirements.txt` file):

```
pip install GRID_LRT
```

1.2 Via Git or Download

Download the latest version from `https://www.github.com/apmechev/GRID_LRT`. To install, use

```
python setup.py build
python setup.py install
```

In the case that you do not have access to the python system libraries, you can use `--prefix=` to specify install folder. For example if you want to install it into a folder you own (say /home/apmechev/software/python) use the following command:

```
python setup.py build
python setup.py install --prefix=${HOME}/software/python
```

Note: NOTE: you need to have your pythonpath containing

"${HOME}/software/python/lib/python[2.6|2.7|3.4]/site_packages"

and that folder needs to exist beforehand or setuptools will complain

CHAPTER 2

Tokens

The GRID_LRT.Token module is responsible for interactions with CouchDB using the PiCaS token framework. It contains a **Token_Handler** object which manages a single _design document on CouchDB, intended for a set of jobs that are logically linked together. In the LOFAR Surveys case, this holds the jobs of a single Observation. Additionally a **Token_Set** object can create batch tokens, upload attachments to them in bulk and change Token fields in bulk as well. This module is used in combination with the *srmlist* class to automatically create sets of jobs with N files each.

2.1 token.py

Location: GRID_LRT/token.py

Imports:

```
>>> from GRID_LRT.token import Token, TokenBuilder #Abstract classes
>>> from GRID_LRT.token import caToken, TokenJsonBUilder, TokenList, TokenView
>>> from GRID_LRT.token import TokenSet
```

Usage:

```
>>> #Example creation of a token of token_type 'test'
>>> from GRID_LRT.auth.get_picas_credentials import picas_cred
>>> pc=picas_cred() #Gets picas_credentials
>>>
>>> from cloudant.client import CouchDB
>>> from GRID_LRT.token import caToken #Token that implements the cloudant interface
>>> from GRID_LRT.token import TokenList
>>> client = CouchDB(pc.user,pc.password, url='https://picas-lofar.grid.surfsara.
↪nl:6984',connect=True)
>>> db = client[pc.database]
>>> 'token_id' in db # Checks if database includes the token
>>> db['token_id'] #Pulls the token
>>> tl = TokenList(database=db, token_type='token_type') #makes an empty list
>>> tl.add_view(TokenView('temp',"doc.type == \"{}\" ".format(tl.token_type))) # adds␣
↪a view to the token
```

(continues on next page)

(continued from previous page)

```
>>> for token in tl.list_view_tokens('temp'):
        tl.append(caToken(token_type=tl.token_type, token_id= token['_id'],␣
↪database=db))
# adds every token in view to the local list (only their ids)
>>> tl.fetch() #Fetch actual data for token in list
>>> t1 = caToken(database=db, token_type='restructure_test', token_id=token_id) #make␣
↪a token (locally)
>>> t1.build(TokenJsonBuilder('/path/to/token/data.json'))
>>> t1.save() #upload to the database
>>> t1.add_attachment(attachment_name='attachment_name_in_db.txt',filename='/path/to/
↪attachment/file') #Adds attachment to token
```

2.1.1 Tokens

class GRID_LRT.token.**Token**(*token_type*, *token_id=None*, ***kwargs*)

__init__(*token_type*, *token_id=None*, ***kwargs*)
x.__init__(…) initializes x; see help(type(x)) for signature

add_attachment()

build(*token_builder*)

reset()

synchronize(*db*, *prefer_local=False*, *upload=False*)
Synchronizes the token with the database.

2.1.2 TokenSet

class GRID_LRT.token.**TokenSet**(*th=None*, *tok_config=None*)
The TokenSet object can automatically create a group of tokens from a yaml configuration file and a dictionary. It keeps track internally of the set of tokens and allows users to batch attach files to the entire TokenSet or alter fields of all tokens in the set.

__init__(*th=None*, *tok_config=None*)
The TokenSet object is created with a TokenHandler Object, which is responsible for the interface to the CouchDB views and Documents. This also ensures that only one job type is contained in a TokenSet.

Args:

param th The TokenHandler associated with the job tokens

type th GRID_LRT.Token.TokenHandler

param tok_config Location of the token yaml file on the host FileSystem

type tok_config str

raises AttributeError, KeyError

add_attach_to_list(*attachment*, *tok_list=None*, *name=None*)
Adds an attachment to all the tokens in the TokenSet, or to another list of tokens if explicitly specified.

add_keys_to_list(*key*, *val*, *tok_list=None*)

`create_dict_tokens`(*iterable={}*, *id_prefix='SB'*, *id_append='L000000'*, *key_name='STARTSB'*, *file_upload=None*)

A function that accepts a dictionary and creates a set of tokens equal to the number of entries (keys) of the dictionary. The values of the dict are a list of strings that may be attached to each token if the 'file_upload' argument exists.

Args:

param iterable The dictionary which determines how many tokens will be created.

The values are attached to each token :type iterable: dict :param id_append: Option to append the OBSID to each Token :type id_append: str :param key_name: The Token field which will hold the value of the dictionary's keys for each Token :type key_name: str :param file_upload: The name of the file which to upload to the tokens (typically srm.txt) :type file_upload: str

`tokens`

`update_local_tokens`()

CHAPTER 3

Staging Modules

These modules are located in GRID_LRT.Staging and can be used to batch stage or check the status of the files on the GRID Storage.

3.1 GRID_LRT.Staging.srmlist

`GRID_LRT.Staging.srmlist.`**`count_files_uberftp`**(*directory*)

`GRID_LRT.Staging.srmlist.`**`make_srmlist_from_gsiftpdir`**(*gsiftpdir*)

`GRID_LRT.Staging.srmlist.`**`slice_dicts`**(*srmdict*, *slice_size=10*)

Returns a dict of lists that hold 10 SBNs (by default). Missing Subbands are treated as empty spaces, if you miss SB009, the list will include 9 items from SB000 to SB008, and next will start at SB010

`class` `GRID_LRT.Staging.srmlist.`**`srmlist`**(*check_OBSID=True*, *check_location=True*, *link=None*)

Bases: `list`

The srmlist class is an extension of Python lists that can hold a list of srm links to data on GRID Storage (LOFAR Archive, Intermediate Storage, etc).

In addition to the regular list capabilities, it also has internal checks for the location and the OBSID of the data. When a new item is appended, these checks are done automatically. Checking OBSID is an optional argument set to True by default.

`__init__`(*check_OBSID=True*, *check_location=True*, *link=None*)

__init__: Initializes the srmlist object.

Parameters

- **`check_OBSID`** (*`Boolean`*) – Boolean flag to check if each added link has the same OBSID
- **`check_location`** (*`Boolean`*) – Boolean flag to check if all files are in the same location (for staging purposes)
- **`link`** (*`str`*) – append a link to the srmlist at creation

append(*item*)
L.append(object) – append object to end

check_location(*item*)

check_str_location(*item*)

count(*value*) → integer – return number of occurrences of value

extend()
L.extend(iterable) – extend list by appending elements from the iterable

gfal_links()
Returns a generator that can be used to generate links that can be staged/stated with gfal

gfal_replace(*item*)
For each item, it creates a valid link for the gfal staging scripts

gsi_links()
Returns a generator which can be iterated over, this generator will return a set of gsiftp:// links which can be used with globus-url-copy and uberftp

gsi_replace(*item*)

http_links()
Returns a generator that can be used to generate http:// links that can be downloaded using wget

http_replace(*item*)

index(*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

insert()
L.insert(index, object) – insert object before index

pop([*index*]) → item – remove and return item at index (default last).
Raises IndexError if list is empty or index is out of range.

remove()
L.remove(value) – remove first occurrence of value. Raises ValueError if the value is not present.

reverse()
L.reverse() – reverse *IN PLACE*

sbn_dict(*pref='SB'*, *suff='_'*)
Returns a generator that creates a pair of SBN and link. Can be used to create dictionaries

sort()
L.sort(cmp=None, key=None, reverse=False) – stable sort *IN PLACE*; cmp(x, y) -> -1, 0, 1

srm_replace(*item*)

stringify_item(*item*)

trim_spaces(*item*)
Sometimes there are two fields in the incoming list. Only take the first as long as it's fromatted properly

3.2 GRID_LRT.Staging.stage_all_LTA

GRID_LRT.Staging.stage_all_LTA.**get_stage_status**(*stageid*)

GRID_LRT.Staging.stage_all_LTA.**location**(*filename*)

`GRID_LRT.Staging.stage_all_LTA.`**`main`**(*filename*, *test=False*)

`GRID_LRT.Staging.stage_all_LTA.`**`process`**(*urls*, *repl_string*, *match*, *test=False*)

`GRID_LRT.Staging.stage_all_LTA.`**`process_surl_line`**(*line*)
Used to drop empty lines and to take the first argument of the srmfile (the srm:// link)

`GRID_LRT.Staging.stage_all_LTA.`**`replace`**(*file_loc*)

`GRID_LRT.Staging.stage_all_LTA.`**`return_srmlist`**(*filename*)

`GRID_LRT.Staging.stage_all_LTA.`**`state_dict`**(*srm_dict*)

`GRID_LRT.Staging.stage_all_LTA.`**`strip`**(*item*)

3.3 GRID_LRT.Staging.state_all

Python module to check the state of files using gfal and return their locality # == # # author: Ron Trompert <ron.trompert@surfsara.nl> – SURFsara # # helpdesk: Grid Services <grid.support@surfsara.nl> – SURFsara # # # # usage: python state.py # # description: # # Display the status of each file listed in "files". The paths # # should have the '/pnfs/. . . ' format. Script output: # # ONLINE: means that the file is only on disk # # NEARLINE: means that the file in only on tape # # ONLINE_AND_NEARLINE: means that the file is on disk # # and tape # # == #

`GRID_LRT.Staging.state_all.`**`check_status`**(*surl_link*, *verbose=True*)
Obtain the status of a file from the given surl.

Args:

param surl the SURL pointing to the file.

type surl str

parame verbose print the status to the terminal.

type verbose bool

Returns:

(filename, status) a tuple containing the file and status as stored in the 'user.status' attribute.

`GRID_LRT.Staging.state_all.`**`check_status_file`**(*surl_list*)
Unimplemented task

`GRID_LRT.Staging.state_all.`**`load_file_into_srmlist`**(*filename*)
Helper function that loads a file into an srmlist object (will be added to the actual srmlist class later)

`GRID_LRT.Staging.state_all.`**`main`**(*filename*, *verbose=True*)
Main function that takes in a file name and returns a list of tuples of filenames and staging statuses. The input file can be both srm:// and gsiftp:// links.

Args:

param filename The filename holding the links whose have to be checked

type filename str

param verbose A toggle to turn off printing out the status of each file.

True by default will print everything out :type verbose: bool

Returns:

ret results A list of tuples containing the file_name and the State

Usage:

```
>>> from GRID_LRT.Staging import state_all
>>> filename='/home/apmechev/GRIDTOOLS/GRID_LRT/GRID_LRT/tests/srm_50_sara.txt'
>>> results=state_all.main(filename)
>>> results=state_all.main(filename, verbose=False)
>>> results[0]
('L229507_SB150_uv.dppp.MS_f6fc7fc5.tar', 'ONLINE_AND_NEARLINE')
```

`GRID_LRT.Staging.state_all.`**`percent_staged`**(*results*)

Takes list of tuples of (srm, status) and counts the percentage of files that are staged (0->1) and retunrs this percentage as float

Usage:

```
>>> from GRID_LRT.Staging import state_all
>>> filename='/home/apmechev/GRIDTOOLS/GRID_LRT/GRID_LRT/tests/srm_50_sara.txt'
>>> results=state_all.main(filename, verbose=False)
>>> state_all.percent_staged(results)
```

3.4 GRID_LRT.Staging.stager_access

It uses an xmlrpc proxy to talk and authenticate to the remote service. Your account credentials will be read from the awlofar catalog Environment.cfg, if present or can be provided in a .stagingrc file in your home directory.

!!Please do not talk directly to the xmlrpc interface, but use this module to access the provided functionality. !! This is to ensure that when we change the remote interface, your scripts don't break and you will only have to upgrade this module.

`GRID_LRT.Staging.stager_access.`**`stage`**(*surls*)

Stage list of SURLs or a string holding a single SURL

Parameters **`surls`** (*`either a list() or a str()`*) – Either a list of strings or a string holding a single surl to stage

Returns An integer which is used to refer to the stagig request when polling

the API for a staging status

`GRID_LRT.Staging.stager_access.`**`get_status`**(*stageid*)

Get status of request with given ID

Args:

param stageid The id of the staging request which you want the status of

type stageid int

Returns:

status A string describing the staging status: 'new', 'scheduled',

'in progress' or 'success'

`GRID_LRT.Staging.stager_access.`**`get_surls_online`**(*stageid*)

Get a list of all files that are already online for a running request with given ID

`GRID_LRT.Staging.stager_access.`**`get_srm_token`**(*stageid*)

Get the SRM request token for direct interaction with the SRM site via Grid/SRM tools

`GRID_LRT.Staging.stager_access.`**`reschedule`**`(`*stageid*`)`
Reschedule a request with a given ID, e.g. after it was put on hold due to maintenance

`GRID_LRT.Staging.stager_access.`**`get_progress`**`()`
Get a detailed list of all running requests and their current progress. As a normal user, this only returns your own requests.

`GRID_LRT.Staging.stager_access.`**`get_storage_info`**`()`
Get storage information of the different LTA sites, e.g. to check available disk pool space. Requires support role permissions.

CHAPTER 4

Sandbox Module

The Sandbox module creates a tar archive of the scripts to be distributed to the worker nodes at the launch of a PiCaS job. The location of the sandbox is stored in the PiCaS token and upon launch, it is downloaded and extracted. The sandbox is created from a configuration file which defines its name, location scrts repository and any additional processing scripts, such as prefactor.

4.1 GRID_LRT.sandbox

CHAPTER 5

Error Codes

Here are a list of errors that the GRID_Sandbox or GRID_Launcher return when processing data on a worker node. The error code is saved in the ‘output’ field of the PiCaS token.

-2 -> Sandbox downloaded but size 0kB
-1 ->
0 -> RUN OK!
1 -> One of Token=${TOKEN}, Picas_usr=${PICAS_USR}, Picas_db=${PICAS_DB} not set
2 ->
3 -> Parset doesn’t exist
4 -> $JOBDIR doesn’t exist
5 ->
6 ->
7 ->
8 ->
9 ->
10 -> Softdrive not found
11 -> LOFAR env cannot be found by GRID_PiCaS_Launcher
12 -> No init_env script
13 ->
14 ->
15 ->
16 ->
17 ->
18 ->
19 ->
20 -> No download File Present
21 -> Download fails
22 -> Data not staged

23 -> pref_cal1 solutions do not download/extract
24 ->
25 ->
26 ->
27 ->
28 ->
29 ->
30 -> No files in uploads folder
31 -> Upload to gsiftp fails
32 -> Upload to gsiftp fails: Pools full!
33 -> Upload to gsiftp fails: File already exists
34 -> Upload to gsiftp fails: File cannot be found (Parent folder not exist?)
35 ->
36 ->
37 ->
…
…
…
90 -> genericpipeline.py stdout file cannot be found!
91 ->
92 ->
93 ->
94 ->
95 ->
96 -> Files not downloaded fully
97 -> dppp memory error in prefactor
98 -> Bad_alloc error in prefactor
99 -> Generic Prefactor Failure

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index