



HAL
open science

The Morumotto / Sicaqo project

Olivier Geber, Constanza Pardo, Jean-Marie Saurel, Arnaud Lemarchand,
Philippe Kowalski, Nicolas Leroy, El-Madani Aissaoui, Claudio Satriano

► **To cite this version:**

Olivier Geber, Constanza Pardo, Jean-Marie Saurel, Arnaud Lemarchand, Philippe Kowalski, et al..
The Morumotto / Sicaqo project. Rencontres scientifiques et techniques RESIF 2019, Nov 2019,
Biarritz, France. hal-02446026

HAL Id: hal-02446026

<https://hal.science/hal-02446026v1>

Submitted on 20 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Abstract : Morumotto (former SiQaCo project) is a tool that automatically fills a seismic data archive, corrects overlaps and requests data to fill the gaps. Data are regularly fetched from a pool of different sources. It also performs data quality control. This software is to be used by network operators.

The software was designed to be able to integrate new sources plugins (to be able to fit any kind of Seismic Network) and new data format/structure in the future.

Beta version available

► Source code available on GitHub (*) :

github.com/IPGP/morumotto



Work flow

Data update

- Read final archive, create gaps index
- Get source inventories
- Create requests from a pool of sources
- When stack is ready, execute it as multithreaded background task
- Clean data, remove overlaps, then merge it to final archive (see below : request algorithm)
- Update final archive statistics

Stack management

- Each source has a priority, data coming from higher priority source will be favoured
- Limitations in local or distant CPU and bandwidth can be modified to fit networks specificities
- Tasks are handled either manually or running automatically in a **crontab**

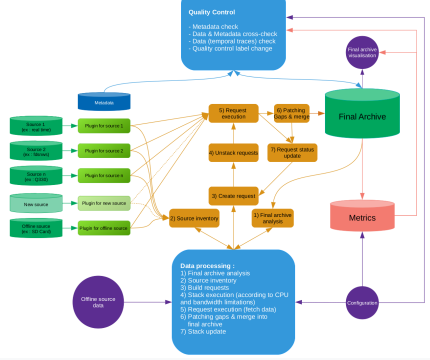
Request algorithm

1. Morumotto read gaps in final archive

FOR EACH GAP :

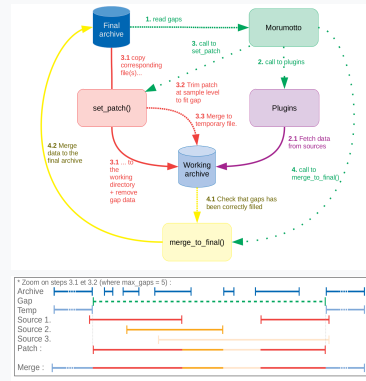
- Encapsulated in a request
- Morumotto calls plugins
 - 2.1 plugins fetches data corresponding to the gap (-patch) from several sources and copy them into the working archive in a temporary directory linked to the request
- Morumotto calls set_patch()
 - 3.1 set_patch() copies files concerned with the gap from the final archive to the working archive, then tries them to remove potential data between gap start and end (in case several gaps have been merged into a bigger one)
 - 3.2 set_patch() merges patched data, given more importance to higher priority source data, into a final patch that fits on sample level the gap
 - 3.3 Merge final patch to temporary file(s) (copied in step 3.1)
 - In case of failure : delete temporary data (unless debug_mode is ON), request status is set to « failed »**
- Morumotto calls merge_to_final()
 - 4.2 merge_to_final compares temporary file(s) with the corresponding file(s) in final archive and checks that no new gap nor data compression problem has been introduced
 - 4.3 merge_to_final overwrites final archive file(s) with the temporary file(s)
 - In case of failure : delete temporary data (unless debug_mode is ON), request status is set to « failed »**

5. Morumotto cleans requests



Quality control

- Check metadata
- Check data quality
- Check waveforms vs metadata
- Check station location
- Plot instrument response
- Plot data completion



Software

Design

- Object oriented
- Modular:
 - New plugin integration is easy
 - New data & metadata formats can be added
- Works indifferently from data format & structure
- Currently supports miniSEED, dataless SEED and stationXML (FDSN)
- For evenly sampled generic timeseries including-but not only-seismic data

Implementation

- Built with Django framework, which includes a template system for the frontend and an ORM for the backend and database management
- Supported RDBMS: MySQL, PostgreSQL
- Algorithms for request creation and stack management are in Python, data quality control and data processing are done using the Obspy library
- Frontend includes several Javascript libraries (Cal-heatmap, Databbles, Leaflet...)
- Multithreading and background tasks is done with Celery, which has a specific django plugin
- Stack monitoring is done with Flower, a Celery monitoring tool
- Source plugin scripts are written in Shell

Daemon, crontab, configuration

- The software is daemonized, so that it runs on the startup of the system (using supervisor)
- It currently uses the **crontab** to execute according to a frequency defined by the user
- *install.sh* script provided, tested in **Debian 9** and **Ubuntu > 16.04**
- First use of the software comes with an initialisation wizard

License

► This program is free and open source, under the GNU General Public License version 3

Command line interface

All data processing & configuration can be done by **command line**. Few examples :

► Create a request manually :

```
>$ python manage.py create_request starttime 2018-04-22T06:00:00Z endtime 2018-04-22T07:00:00Z nslc_list PF.PRO.00.HHZ PP.PRO.00.HHZ --source_list FDSNW3_IPGP LOCAL_DIR_HOME
```

► Execute stack manually

```
>$ python manage.py exec_stack
```

► Update data for a specific window of time

```
>$ python manage.py window_update --window_starttime 2018-04-22T00:00:00Z window_endtime 2018-04-28T00:00:00Z
```

Plugins

- New plugins can be easily integrated into the software
- FDSN Webservice plugin code :

```
230 # Read arguments
231 opts=(getopt \
232 --longoptions "sprint" "slist" "--(ARGUMENT_LIST001)" \
233 --name "fsdnwebservice" "url" \
234 --options "" \
235 --)
236
237 eval set -- "${opts}"
238
239 # Extract options and their arguments into variables.
240 while true; do
241   case "$1" in
242     --url) ONLINE_FLAGS+=url; CLIENT+=url; shift 2 ;;
243     --postfile) POSTFILE+=url; shift 2 ;;
244     --workspace) WORKSPACE+=url; shift 2 ;;
245     --data-format) DATA_FORMAT+=url; shift 2 ;;
246     --blocksize) BLOCKSIZE+=url; shift 2 ;;
247     --compression) COMPRESS+=url; shift 2 ;;
248     --connect-infos) CONNECT_INFOS+=url; shift 2 ;;
249     --log-level) _VERBOSITY+=url; shift 2 ;;
250     --) shift; break ;;
251     *) echo "Wrong call to the script" usage; exit 1 ;;
252   esac
253 done
```

- On the left : I/O (always the same),
- On the right : the part that needs to be modified to fetch data from a new source
- Then create a new class in *source.py* :

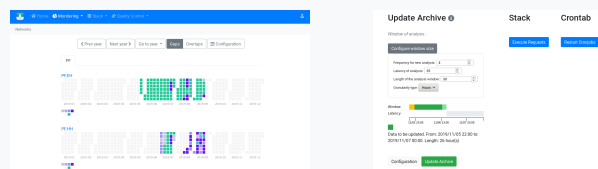
```
213 class FdswnSourcePlugin:
214     ...
215     Class for the FDSN Web Service source plugin
216     The plugin is plugins/fdswn.sh
217     ...
218     plugin_file = "fdswn.sh"
219     webservice_file = "fdswn_webservice.py"
220     template = "client_url_(url:service:fsdn.edu)"
221     ...
222     def set_connect_infos(self, parameters, limit_rate):
223         ...
224         returns a string containing the infos to connect to the fdswn.sh plugin
225         from the parameters and limit rate
226         ...
227         return "client_url_limit_rate={parameters}_limit_rate"
```

Yes, that's the full class definition

Interface

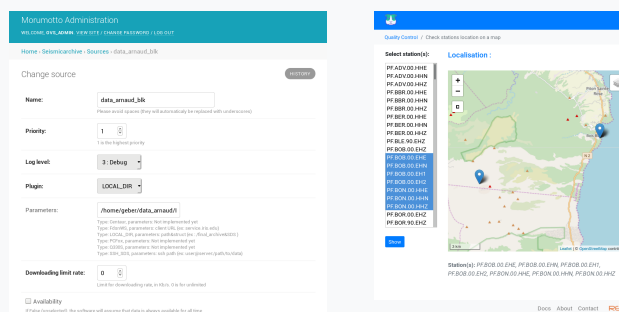
Web interface

- Morumotto is running on a **server**, users can access it with a classical **web browser**
- Administration interface for configuration
- Different **user levels** : normal, advanced and admin
 - Normal user can monitor network gaps, overlaps and perform quality control (not affecting data)
 - Advanced user can create data requests, change update settings and statistics
 - Admin can configure, add & delete requests, statistics, users, etc.

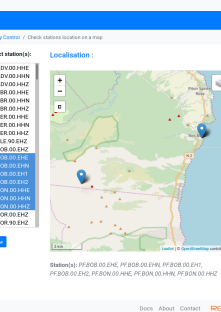


Monitoring interface : daily gaps per component view

Configuration



Administration panel : source plugin configuration



QC interface : station location view

Development

Calendar

- Functional specification & design dossier shared with RESIF community in April & September 2018
- Available on github :
 - github.com/IPGP/siqaco/blob/master/CDC_ValidationDonnees.pdf
 - github.com/IPGP/siqaco/blob/master/SiQaCo-Dossier-de-conception.pdf
- Development from Sep. 2018, involving RESIF, volcano observatories from IPGP community & Geoscop

Perspectives

- (*) Release candidate version will be pushed to the RESIF GitHub
- Source plugins currently under development : Centaur, Q330, NAQS
- Quality Control currently under development
- Installation & testing in observatories & RESIF
- API & Documentation in progress