Morumotto

MOdular a**R**chive b**U**ilder from
Multiple **O**rigin **T**emporal **T**races
(& **O**ther stuff)

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

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 n**etwork 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

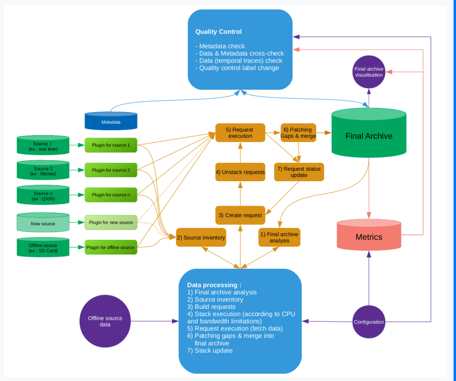
Work flow

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**



Quality control

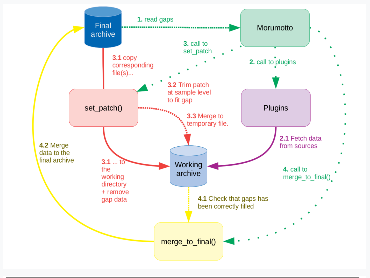
► Check metadata
► Check data quality
► Check waveforms vs metadata
► Check station location
► Plot instrument response
► Plot data completion

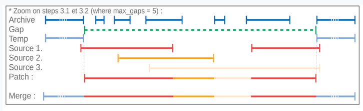
Request algorithm

```
1. Morumotto read gaps in final archive

FOR EACH GAP :
        ------------Encapsulated in a request--------------
        2. 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
        3. Morumotto calls set_patch()
                3.1 set_patch() copies files concerned with
                the gap from the final archive to the working
                archive, then trims 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, givin
                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 »
        4. 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 »
        -------------------End of request----------------------

5. Morumotto cleans requests
```

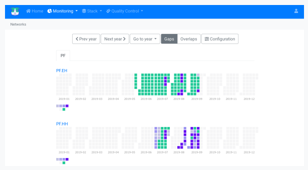

Final archive
1. read gaps
Morumotto
3. call to set_patch
2. call to plugins
3.1 copy corresponding file(s)...
3.2 Trim patch at sample level to fit gap
set_patch()
Plugins
3.3 Merge to temporary file.
4.2 Merge data to the final archive
2.1 Fetch data from sources
3.1 ... to the working directory + remove gap data
Working archive
4. call to merge_to_final()
4.1 Check that gaps has been correctly filled
merge_to_final()



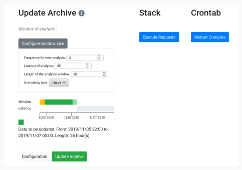
Interface

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.

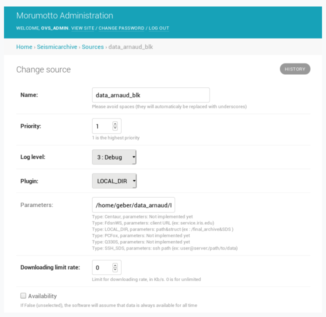


Monitoring interface : daily gaps per component view



Window of analysis :
Frequency for new analysis: 4
Latency of analysis: 35
Length of the analysis window: 30
Granularity type: Hours
Data to be updated. From: 2019/11/05 22:00 to: 2019/11/07 00:00. Length: 26 hour(s)

Configuration


Morumotto Administration
WELCOME, OVS_ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Seismicarchive › Sources › data_arnaud_blk
HISTORY

Change source

Name: data_arnaud_blk
Please avoid spaces (they will automaticaly be replaced with underscores)

Priority: 1
1 is the highest priority

Log level: 3 : Debug

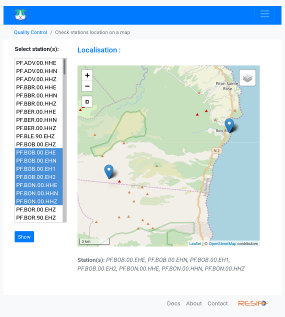
Plugin: LOCAL_DIR

Parameters: /home/geber/data_arnaud/l
Type: Centaur, parameters: Not implemented yet
Type: FdsnWS, parameters: client URL (ex: service.iris.edu)
Type: LOCAL_DIR, parameters: path&struct (ex : /final_archive&SDS)
Type: PCFox, parameters: Not implemented yet
Type: Q330S, parameters: Not implemented yet
Type: SSH_SDS, parameters: ssh path (ex: user@server:/path/to/data)

Downloading limit rate: 0
Limit for downloading rate, in Kb/s. 0 is for unlimited

☐ Availability
If False (unselected), the software will assume that data is always available for all time

Administation panel : source plugin configuration


Quality Control / Check stations location on a map
Select station(s):
Localisation :
Show
Piton Sainte-Rose
Bois-Bl
N 2
3 km
Leaflet | © OpenStreetMap contributors
Docs
About
Contact

PF.ADV.00.HHE
PF.ADV.00.HHN
PF.ADV.00.HHZ
PF.BBR.00.HHE
PF.BBR.00.HHN
PF.BBR.00.HHZ
PF.BER.00.HHE
PF.BER.00.HHN
PF.BER.00.HHZ
PF.BLE.90.EHZ
PF.BOB.00.EHZ
PF.BOB.00.EHE
PF.BOB.00.EHN
PF.BOB.00.EH1
PF.BOB.00.EH2
PF.BON.00.HHE
PF.BON.00.HHN
PF.BON.00.HHZ
PF.BOR.00.EHZ
PF.BOR.90.EHZ

Station(s): *PF.BOB.00.EHE, PF.BOB.00.EHN, PF.BOB.00.EH1, PF.BOB.00.EH2, PF.BON.00.HHE, PF.BON.00.HHN, PF.BON.00.HHZ*

QC interface : station location view

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*, *Datatables*, *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 deamonized, 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.HH? PF.RER.90.HHZ --source_list FDSNWS_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 :

```
# Read arguments
opts=$(getopt \
    --longoptions "$(printf "%s::," "${ARGUMENT_LIST[@]}")" \
    --name "$(basename "$0")" \
    --options "" \
    -- "$@"
    )
eval set --${opts}
# extract options and their arguments into variables.
while true; do
    case "$1" in
        --is_online) ONLINE_FLAG=true; CLIENT=$2; shift 2 ;;

        --postfile) POSTFILE=$2; shift 2 ;;

        --workspace) WORKSPACE=$2; shift 2 ;;

        --data-format) DATA_FORMAT=$2; shift 2 ;;

        --blocksize) BLOCKSIZE=$2; shift 2 ;;

        --compression) COMPRESS=$2; shift 2 ;;

        --connect-infos) CONNECT_INFOS=$2; shift 2 ;;

        --log-level) __VERBOSE=$2; shift 2 ;;

        --) shift ; break ;;
        *) echo "Wrong call to the script!" usage ; exit 1 ;;
    esac
done
```

```
# Get client name
CLIENT=$(echo ${CONNECT_INFOS} | awk -F '[:?=&]' '{print $2}')
.log 3 "client: "${CLIENT}
LIMIT_RATE=$(echo ${CONNECT_INFOS} | awk -F '[:?=&]' '{print $4}')
if [ "${LIMIT_RATE}" = "" ]; then
  LIMIT_RATE="0k"
fi
.log 3 "Limit rate: ${LIMIT_RATE}"

now="$(date +'%s')"

FILENAME=$(mktemp ${WORKSPACE}/XXXXXX.ws_wget.${DATA_FORMAT})
.log 3 "Filename: ${FILENAME}"

wget -q --post-file=${POSTFILE} --limit-rate=${LIMIT_RATE} \
-O ${FILENAME} "http://${CLIENT}/fdsnws/dataselect/1/query" \
2>&1 >/dev/null || exit_wget=$? ;

EXIT_STATUS=$(convert_exit ${exit_wget})
.log 3 "Exit status :" ${EXIT_STATUS}

if [ "${EXIT_STATUS}" -ne 0 ]; then
  .log 1 ${EXIT_STATUS}
  exit ${EXIT_STATUS};
fi

.log 3 "WGET success"

if ! ( [ -f ${FILENAME} ] && [ -s ${FILENAME} ] ) ; then
  .log 1 "No data found"
  exit 2;
fi

correct_seed.sh --log-level=${__VERBOSE} \
  --input-file=${FILENAME} \
  --output-dir=${WORKSPACE} \
  --blocksize=${BLOCKSIZE} \
  --encoding=${COMPRESS}

.log 3 "FDSNWS Script finished with success"
exit 0
```

► 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* :

```
class FdsnWS(SourcePlugin):
    """
    Class for the FDSN Web Service source plugin
    The plugin is plugins/fdsnws.sh
    """
    plugin_file = "fdsnws.sh"
    avail_plugin_file = "fdsnws_availability.sh"
    template = "client URL (ex: service.iris.edu)"

    def set_connect_infos(self,parameters,limit_rate):
        """
        returns a string containing the infos to connect to the fdsnws.sh plugin
        from the parameters and limit rate
        """
        return "client:%s?limit-rate=%sk" % (parameters,limit_rate)
```

*Yes, that's the **full** class definition*

Development

Calendar

► Functional specicication & 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 & Geoscope

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