mqrun Documentation

Release 0.1

Adrian Seyboldt

May 19, 2014

Contents

MaxQuant is a quantitative proteomics software package that unfortunatly only runs on Windows and is designed for use with a graphical user interface. Both make it hard to integrate into a larger workflow.

mqrun consists of four parts, that try to mitigate those problems:

mqrun.mqparams This is a small library that converts user supplied json-parameter files into the rather peculiar configuration files of MaxQuant and supplies a small helper function that calles the MaxQuant executable with this configuration. The format of the parameter file is documented in *Parameter file format*. Since the format of the MaxQuant configuration file keeps changing, mqparams supports one version of MaxQuant only.

mqrun.fscall `fscall` is a library that handles requests to another machine, where the filesystem is the only communication channel. It provides status messages, simple error handling, access to logfiles and a heartbeat that tells the client that the server is still working on a request.

mqrun.mqdaemon `mqdaemon` uses `fscall` to provide a server that can run on a Windows machine and handles requests for MaxQuant. It includes basic load balancing. See `mqdaemon -h` for options.

mqrun.mqclient `mqclient` is a small client library that wraps the `fscall` routines to provide a convenient interface for programs running on linux (or windows) machines, that want to run MaxQuant.

If configured correctly it can also start a virtual machine running Windows and use that to run MaxQuant without the need for a dedicated Windows machine.

CHAPTER 1

Contents

1.1 Quickstart

1.1.1 Calling MaxQuant locally

blubb.

1.1.2 Calling MaxQuant on a dedicated Windows machine

Suppose the two hosts `win_host` and `linux_host` are in a network and we would like to call MaxQuant on the `win_host`, but control the process and provide the input and parameter files from the linux machine.

In this tutorial I will assume that MaxQuant is already installed on `win_host` and that a share has been setup: `/mnt/win_share` should be accessable from `win_host` as `Z:\\`. There are countless tutorials for this setup, but you can find a quick description for this in (**:todo:'not yet'**).

First we need to install python3.3 or later on the windows machine. Download the msi installer from the Python download page and install Python. For ease of use, the option to add the executables to the PATH should be enabled. (**:todo:'check that this option exists...'**)

Next, install `mqrun` on both machines:

```
pip install mqrun
```

On linux this needs administrator priviliges, although a local installation with virtualenv is possible.

Note: python2.7 is ok on linux, the windows version needs at least python3.3

Start the `mqrun.mqdaemon` server process on the windows machine:

```
cd Z:
mkdir requests
mqdaemon -h  # have a look at the options...
mqdaemon --mqpath C:\\path\to\MaxQuant\dir --logfile maxquant.log requests
```

You can now run MaxQuant from linux (using python):

```
from mqrun import mqclient
import json

with open('paramfile.json') as f:
    params = json.load(f)
```

```
path_data = {
    "raw_file1": "/path/to/raw/file",
    "raw_file2": "/path/to/raw/file2",
    "fasta_file": "/path/to/fasta/file",
}

maxquant = mqclient.mqrun(params, path_data, share='/mnt/win_share_requests')

maxquant.wait()
try:
    outfiles = maxquant.result()
except TimeoutError:
    print("Connection lost or server overloaded")
except Exception as e:
    print("Error executing MaxQuant: " + str(e))
else:
    print(outfiles)

print("Logfile\n=======\n")
print(maxquant.log)  # print the logging output of the server
```

The format of the parameter file is explained in `mqrun.mqparams`.

1.1.3 Use MaxQuant on an virtual machine

Requirements:

- qemu with kvm enabled on the linux machine
- guestfs + python wrappers on linux
- A windows virtual machine image

Configure the windows machine

Start the windows machine with qemu and the software image at (TODO)

```
qemu -hda winvm.img -boot c -hdb mqrun_image.img -enable-kvm -m 1024
```

and execute the file `install.bat` on `d:\\`.

Execute MaxQuant exactly as in the example above, but replace the keyword argument `share` in the `mqclient.mqrun` call by `img=path/to/win/image`.

1.2 Installation

1.2.1 Windows setup

The newest version of MaxQuant (1.5.0.0) requires a 64 bit system with Windows A 64Bit machine is preferred by MaxQuant.

Make sure `MaxQuant 1.4.1.2`, `.NET 4.5` and `MSFileReader` are installed.

Download python 3.4 or newer from python.org (the Windows x86-64 MSI installer). Make sure the python executables are added to the path.

Open a terminal and execute

```
pip install mqrun
```

Todo

Get rid of those error dialogs!

Add fasta files to Andromeda?

1.2.2 Linux setup

1.2.3 Configure samba

1.3 Parameter file format

`mqrun` parameter files can be json or yaml formatted. To keep this tutorial easy to read we will only use yaml files. Just replace all yaml file endings by `.json` and use the standard library json module to load the data if you want to use json instead.

`mqrun` parameter files must contain the sections `rawFiles`, `fastaFiles` and `globalParams`. The sections `topLevelParams` (will probably move to `globalParams` soon) and `MSMSParams` are optional.

The sections `globalParams`, `topLevelParams`, `MSMSParams` and the `params` subsection of `rawFiles` must contain a `defaults` value. This will set default values for all other parameters in this section. The output of `mqrun` includes a file with all parameters explicitly set so you can see what values have been used.

1.3.1 Global parameters

This section corresponds to the `global` tab in the MaxQuant GUI and includes parameters that are not specific to a particular input file. For example

```
globalParams:
  defaults: default
  equalIl: true
  matchBetweenRuns: true
  fixedModifications:
    - "Carbamidomethyl (C)"
```

1.3.2 Raw file section

The parameter files do not include paths to raw data, instead each input file is specified by a name and corresponding parameters. When `mqrun` is called, it expects a mapping from input file names to actual file paths. This way the same configuration can be used for different data sets and on different computers without changing the parameter file.

In theory each input file could use a different set of file specific parameters, but usually a large number of input files share their configuration. Theses groups of input files correspond to the parameter groups in the MaxQuant GUI. The section `rawFiles` is a list of these groups.

Each group has two sections of its own: `files`, a list of names, fractions and experiments, and a parameter section `params`.

A configuration with two parameter groups could look like this:

```
rawFiles:
- files:
    - name: 20090812_SvNa_SA_phospho_2_p1
      fraction: 1
    - name: 20090812_SvNa_SA_phospho_2_p2
      fraction: 2
    - name: 20090812_SvNa_SA_phospho_2_p7
      fraction: 3
    - name: 20090812_SvNa_SA_phospho_2_p8
      fraction: 4
  params:
    defaults: default
    intensityDependentCalibration: true
    isotopeTimeCorrelation: 0.8
    centroidMatchTol: 10
    isotopeMatchTol: 3
    mainSearchTol: 7.5
    variableModifications:
      - "Acetyl (Protein N-term)"
      - "Oxidation (M)"
      - "Phospho (STY)"
    useMs1Centroids: true
    matchType: "NoMatching"
    multiplicity: 2
    maxLabeledAa: 3
    labelMods:
      - ["Arg6", "Lys8"]
      - ["Arg10", "Lys8"]
- files:
    - name: 20090812_SvNa_SA_phospho_2_FT1
      fraction: 5
    - name: 20090812_SvNa_SA_phospho_2_FT2
      fraction: 6
    - name: 20090812_SvNa_SA_phospho_2_FT3
      fraction: 7
    - name: 20090812_SvNa_SA_phospho_2_7
      fraction: 8
    - name: 20090812_SvNa_SA_phospho_2_8
      fraction: 9
    - name: 20090812_SvNa_SA_phospho_2_9
      fraction: 10
    - name: 20090812_StPe_SA_phospho_2_10
      fraction: 11
  params:
    defaults: default
    intensityDependentCalibration: true
    isotopeTimeCorrelation: 0.8
    centroidMatchTol: 10
    isotopeMatchTol: 3
    mainSearchTol: 7.5
    variableModifications:
      - "Acetyl (Protein N-term)"
      - "Oxidation (M)"
      - "Phospho (STY)"
    useMs1Centroids: true
    matchType: "NoMatching"
    multiplicity: 2
    maxLabeledAa: 3
```

```
    labelMods:
      - ["Arg6", "Lys8"]
      - ["Arg10", "Lys8"]
```

1.3.3 Fasta files

The fasta files are specified in the `fastaFiles` section. It must include two subsections, `fileNames` and `firstSearch`. The as with the raw files the actual file paths are set when `mqrun` is executed.

An example:

```
fileNames:
  - uniprot_sprot
firstSearch: []
```

1.4 API Documentation

1.4.1 Convert parameters (`mqrun.mqparams`)

Convert between json or yaml and MaxQuant configuration files.

Contents

`xml_to_data`(xml_tree[, logger])	Extract parameters and file-paths from MaxQuant configuration file
`data_to_xml`(user_data, file_paths, ...[, ...])	Convert parameter set to MaxQuant xml.
`mqrun`(binpath, params, datapaths, outdir, tmpdir)	Run MaxQuant with specified parameters and paths.

mqrun.mqparams.xml_to_data

mqrun.mqparams.**xml_to_data**(*xml_tree*, *logger=None*)

Extract parameters and file-paths from MaxQuant configuration file

Parameters

- **xml_tree** (*xml.etree.ElementTree.Element.Tree*) – MaxQuant configuration file.
- **logger** (*logging.Logger, optional*) – Logger for the conversion process. Use `logging.getLogger('mqparams')` if not specified.

Returns

- **params** (*dict*) – Parameters as described in module doc
- **path_data** (*mqparams.ExtraMQData*) – Path information from xml file
- *extra_data has the following attributes*
- **file_paths** (*dict*) – Mapping from names for input files to paths
- **fasta_paths** (*dict # TODO merge with file_paths*) – Same for fasta files
- **output_dir** (*pathlib.Path*) – Path to the output directory
- **tmp_dir** (*pathlib.Path*) – Path to temporary directory

mqrun.mqparams.data_to_xml

mqrun.mqparams.**data_to_xml**(*user_data*, *file_paths*, *fasta_paths*, *output_dir*, *tmp_dir=None*, *logger=None*)

Convert parameter set to MaxQuant xml.

Parameters

- **user_data** (*dict*) – The parameters for MaxQuant as described the module doc of mqparams.
- **file_paths** (*dict*) – Mapping from names in parameter set to actual file paths.
- **fasta_paths** (*dict # TODO merge with file_paths*) – Mapping from names to file paths.
- **output_dir** (*pathlib.Path*) – Write the output files to this directory.
- **tmp_dir** (*pathlib.Path, optional*) – Base dir for temporary data, needs lots of space. Use system default if not specified.
- **logger** (*logging.Logger, optional*) – Logger for the conversion process. Use `logging.getLogger('mqparams')` if not specified

Returns **params_xml** – Configuration file for use with `MaxQuantCmd.exe`

Return type xml.lxml.ElementTree.ElementTree

mqrun.mqparams.mqrun

mqrun.mqparams.**mqrun**(*binpath*, *params*, *datapaths*, *outdir*, *tmpdir*, *logger=None*)

Run MaxQuant with specified parameters and paths.

Parameters files

Example data:

```
>>> import json
>>> from pathlib import Path
>>> from pprint import pprint

>>> # TODO how about some sensible parameters ;-)
>>> with open('paramfile.json') as f:
>>>     param_file = f.read()

>>> print(param_file)
{
    "rawFiles": [
        {
            "name": "input1",
            "params": {
                "defaults": "default",
                "variableModifications": [
                    "Oxidation (M)",
                ]
            }
        },
        {
            "name": "input2",
            "params": {
                "defaults" :"default",
```

```
            }
        }
    "fastaFiles": {
        "fileNames": ["fasta1"],
        "firstSearch": ["fasta1"],
    }
    "globalParams": {
        "defaults": "default",
        "matchBetweenRuns": True
    }
}
```

```
>>> # load json data
>>> params = json.load(param_file)
```

Each parameter file must contain the sections "rawFiles" and "fastaFiles". "globalParams" and "MSMSParams" are optional.

The input files (raw and fasta) are only identified by a unique name. You must specify the paths for each of the input files in a dictionary and pass that to the relevant functions.

The "params" sections in "rawFiles" and the sections "globalParams" and "MSMSParams" have the optional argument "defaults", that specifies default values for the other parameters in that section.

TODO: write some of those and document how to add more

`mqschema.json` contains a json-schema for this file format along with descriptions for a few of the parameters.

Example

To convert above parameters to xml wee need a mapping to the file paths of the input files. Let's say they are stored in the following locations:

```
>>> paths = {
>>>     "input1": Path("C://data/input1.raw")
>>>     "input2": Path("C://data/input2.raw")
>>>     "fasta1": Path("C://data/fasta1.fasta")
>>> }
```

Convert to xml:

```
>>> outdir = Path('C://output/')
>>> tmpdir = Path('C://tmp')
>>> xmltree = data_to_xml(params, paths, outdir, tmpdir)
```

We can now write that xml file to disk and run MaxQuantCmd on it:

```
>>> xmltree.write(str(outdir / "params.xml"))
```

To convert it back to our json format we do:

```
>>> params_, path_data = xml_to_data(xmltree)
>>> path_data.paths == paths
True
>>> path_data.outdir == outdir
True
>>> path_data.tmpdir == tmpdir
True
```

Since the xml file does not contain information about which default values have been used, it specifies all values and differs from the original. But converting it back to xml should yield the same result as before:

```
>>> from sort_xml_tags import equal_sorted
>>> xmltree_ = data_to_xml(params_, *path_data)
>>> equal_sorted(xmltree_, xmltree)
True
```

1.4.2 MaxQuant Server (`mqrun.mqdaemon`)

Start a daemon that listens for new directories. Each new directory that satisfies some conditions will be interpreted as a request to run MaxQuant on the files in that directory. The protocoll that is used to communicate failure or success is described in *fscall.py*.

Load balancing

This program uses a semaphore to limit the number of instances that run MaxQuant at the same time. If a task can not get access to MaxQuant after some amount of time, its status will be set to FAILED. If a task runs for too long it is canceled.

Security

All output and log files will probabley be readable by any user who has permissions to use this service, so if you need private requests, use a dedicated server process for each user.

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

m