lorax Documentation

Release 0.94.44

Joel Berendzen

Oct 03, 2017

See the following sections for details:

`lorax` is a Flask-based web service that uses asynchronous queues (via RQ) to calculate and serve up multiple sequence alignments and phylogenetic trees. `lorax` uses pre-calculated Hidden-Markov Models (HMM) of protein families together with `hmmalign` to calculate multiple sequence alignments in either peptide or DNA space.

`lorax` calculates phylogenetic trees from either DNA or protein sequences by external processes in a computationally-intensive step that can take from minutes to days depending on the number of sequences and their sizes. Generally the computational time increases linearly with the average length of sequences and as the square of the number of input sequences. The tree-building software that `lorax` knows about are:

Tree Builder	Description
Fast-Tree	FastTree is the fastest tree-builder and is the default tree-building algorithm.
RAxML	RAxML is believed to be the more accurate tree-building algorithm, but at its fastest is probably 100x slower than FastTree. It is possible to use the RaxML EPA algorithm to do placement of test sequences on existing RaxML trees.

CHAPTER 1

Sections

Design Assumptions & Goals

We were not setting out to write the underlying bioinformatics code, but rather to wrap existing code. Because multiple codes are needed, the resulting software would have to be an ecosystem of roughly 7 different codes rather than a single monolithic package.

Python was chosen as the implementation language because that language has a rich set of packages needed for implementation. Python 3 was chosen over Python 2 because of longevity concerns, despite some possible dependencies not yet fully supporting Python 3.

We wished to support three distinct Unix platforms (linux, BSD, and MacOS) for development, testing, and deployment. Deployment would eventually be as a non-privileged, non-login user, preferably via a single rc script. It was desired to keep this rc script as simple as possible because root access for testing in the production environment was limited.

Only working `bash` and C compilers were assumed to be deployment prerequisites; everything else needs to be built as a non-privileged user without depending on a particular package manager.

The primary design goal for lorax was to implement best practices for:

- Service architecture: make use of RESTful endpoints, with a migration path to GraphQL in the near future. A great deal of effort goes into writing (non-scientific) web services these days, so make use of those tools and code.
- Software engineering: write code with low technical debt, suitable for re-use later.
- Testing: use Continuous Integration services and measure code coverage.
- Deployment: deployment via a small number of scripts, use templating to handle the many configuration scripts needed.
- Monitoring: enable error tracking and performance monitoring.

Conceptual Design

An early decision was made to base the service around Flask, because that application environment was powerful with a large set of plug-ins. Django was rejected its strongest advantages (e.g., user authentication) were not needed, making the additional complexity not worth it.

RQ (Redis Queues) was chosen over Celery because RQ is so much simpler and smaller. In the future, we need to implement backfill queueing, and the code base that would have to be modified is much smaller in RQ.

We need to start and monitor many different processes. Supervisord seemed like the best fit for launching and monitoring, despite Python 3 support being not quite ready. We forked supervisord and enabled explicit Python 3 support, with installation direct from repository.

The block diagram (from the HTTP point of view) of processes that run to implement lorax is shown below:

Red lines in this figure represent http connections (whether over IP or unix sockets). The components in this software ecosystem are:

- A front-end web proxy (nginx) to handle dispatching, compression, and security.
- A Flask-based service process (lorax) that handles configuration and dispatching as well as serving up results.
- A fast database (redis) used for asynchronous communications.
- Calculational queues running (aligner, treebuilder) running multiple binaries.
- A process launcher and manager (supervisord).
- A server for performance and usage stats (prometheus).

Planning Your Installation

Installation of lorax under linux can be done either via a *direct install* (using either build scripts we supply or using your distribution's package manager) or an *indirect install* via Anaconda Python and packages in the bioconda channel. Which path you choose depends on your OS and general preferences. If you are installing on BSD, direct installs are your only choice as Anaconda Python is not available for this platform. If you are installing on a Mac, you will need a number of prerequisites that are most easily satisfied via Anaconda Python packages. If you are installing on linux, Anaconda Python will get you up and running more quickly while direct installs are more heavily tested and will give you more control of the installation. This is especially important if you anticipate using RAxML heavily and you wish to take advantage of AVX/AVX2 hardware.

MacOS Installation

`lorax` was tested under MacOS Sierra (10.12.6). The minimum OS version for proper function of Anaconda Python is MacOS 10.12.3.

Installation of `lorax` under MacOS requires use of `git`. To test if you have `git` installed, open a terminal window `(under Applications -> Utilities)` and issue the command:

```
git --version
```

If this produces an error, the easiest way to install git is by installing the XCode command-line tools, part of the XCode utilities, which are available for free (but large) download from the App Store.

Besides the XCode tools, you will also need to install gcc (because the current Mac clang does not include OpenMP) as well as the PCRE and OpenSSL libraries. The tested configuration was with PCRE-8.40 and OpenSSL-1.10.f. Both libraries were configured with `CC=clang` and installed to `/usr/local`.

Direct Installation

You may choose to install dependencies yourself (typically using your distribution's package manager) or via the `lorax_build.sh` build script. The latter is recommended because it has pinned dependencies.

Installation via lorax_scripts.sh

We strongly recommend that you do the installation in a clean directory as a non-privileged user. The suggested location for that directory is ~/.lorax/build. If you agree with this choice, you can start your installation by issuing the following commands:

```
mkdir -p ~/.lorax/build
cd ~/.lorax/build
curl -L -o lorax_tool https://raw.githubusercontent.com/LegumeFederation/lorax/master/
↪build_scripts/lorax_tool.sh
chmod 755 lorax_tool
```

If you wish to customize anything about the installation, you should issue the following command:

```
./lorax_tool create_scripts
```

Then, review and edit the `my_build.sh` script to reflect the directories you wish to use for your installation.

After you are happy with the configuration, run:

```
./lorax_tool build
```

to do the build. After the build completes without error, follow the configuration instructions.

Installation via Package Manager

The following are the dependencies for installing and running lorax, with tested versions shown in parentheses:

Dependency	Version
C compiler	clang (8.1.0 on MacOS and 3.4.1 on FreeBSD) gcc (7.1.0 on linux and 4.8.5 on MacOS). Note that MacOS requires BOTH clang and gcc.
git	Any version supported by GitHub should work.
python	python 3 with numpy (3.6.2, minimum of 3.4.0)
hmmer	3.1b2
raxml	8.2.11
redis	4.0.1
nginx	1.13.4

`lorax` requires a more recent version of setuptools than that distributed with python 3.6, as well as the use of pip. Both of these may break distro package manager assumptions, so you will need to either create a working virtual environment or a root environment in user space.

I prefer to use the name `loraxenv` for the virtual environment to prevent confusion between package and environment names. You should create this environment as the user under which you intend the web server to run, using the python version you wish to use selected as default, with the following commands:

```
python3.6 -m virtualenv loraxenv
cd loraxenv
source bin/activate
```

Make sure that the `python` and `pip` commands run python and pip from the virtual environment; if not, you will need to create symlinks in the virtual environment binary directory.

Next use pip to install dependencies:

```
pip install -U setuptools
pip install -e 'git+https://github.com/LegumeFederation/supervisor.git@4.0.0
↪#egg=supervisor==4.0.0'
pip install -U lorax
```

The lorax_env script is the only executable that you will need to control lorax and associated process. You should usually not put the entire lorax bin/ directory into your path, as this may cause binaries from the virtual environment such as python to be used in contexts where they were not intended to be used. The easiest way to do this is with a symlink to a directory on your path, typically `~/bin`:

```
ln -s /path/to/loraxenv/bin/lorax_env ~/bin
```

You no longer need to work inside the virtual environment you created. Leave the virtual environment with the following command:

```
deactivate
```

Next, follow the configuration instructions.

Anaconda Installation

This section describes installation via Anaconda Python's package manager, `conda`. If you want to do a direct install, you want the previous chapter.

If you don't have Anaconda Python installed, get it by visiting the URL https://www.continuum.io/downloads. `lorax` was tested under version 3.6 of Anaconda Python. As of this writing, there is no compelling reason to use earlier or later versions. You must use a 64-bit version. The command-line installer version is fine, and 25% smaller. When you have downloaded the installer, issue the following command from a terminal window in the Downloads directory:

```
bash Anaconda3-<version>-<platform>.sh
```

where `<version>` is the current version number and `<platform` is your OS. As part of installation, please ensure that the Anaconda Python `bin/` directory is prepended to your PATH (in `~/.bash_profile` on the Mac) or else you will have to prepend it manually before continuing the installation and every time you wish to use `lorax`.

Configure channels

lorax requires some packages that are in non-default channels, which may be obtained by the following:

```
conda config --add channels conda-forge
conda config --add channels defaults
conda config --add channels bioconda
```

After the channels are configured, update the installation by issuing:

```
conda update --all
```

This command usually results in several packages being updated.

Create a virtualenv

It is not advisable to install `lorax` in the root environment because lorax's dependencies could break other packages. As of the current conda version (4.3.25), there are four packages in the channels that are too old for use with lorax:

Package	Anaconda Version	Required Version
setuptools	27.2.0	>30.3.0
markupsafe	0.23	1.0.0
gunicorn	19.1.0	19.7.1
supervisor	3.3.2	4.0 (from git)

Replacing setuptools, in particular, could cause breakage of your root Anaconda Python distribution. For this reason, it is required to create a python virtual environment for lorax and its dependencies. I prefer to name this environment "loraxenv" to prevent confusion between package and environment names. You should create this environment as the user under which you intend the web server to run (using `sudo -u USER` if necessary), using the following commands:

```
conda create -y --name loraxenv biopython click hmmer raxml redis nginx
source activate loraxenv
```

Install gcc if using MacOSX

If you are installing under MacOS (and only under MacOS), gcc and libgcc MUST be installed in the current virtual environment:

```
conda install -y gcc libgcc
```

Failure to do this step will result in a message to the effect "Unable to resolve paths in MacOS executable" in the lorax installation step later.

Upgrade setuptools

Next continue the installation by updating setuptools and installing a special version of supervisord from a git repository. The first will end with an error; this is expected, so don't be worried by it:

```
pip install -U setuptools
pip install -e 'git+https://github.com/LegumeFederation/supervisor.git@4.0.0
↪#egg=supervisor==4.0.0'
```

Choose root location

When lorax installs itself, it will place some files in the `bin/` and `etc/` subdirectories of `LORAX_ROOT`. Normally `LORAX_ROOT` is set to the same directory as parent of the python installation in use (that is, in `sys.prefix`). This is usually fine for most installations, but if you wish to override it, you must define `LORAX_ROOT` before installation:

```
export LORAX_ROOT=some_directory_you_choose
```

Install lorax

Next, install the lorax dependencies and binaries:

```
pip install -U lorax
```

Put lorax_env script in alias or on path

The lorax_env script is the only executable that you will need to control lorax and associated process. You should usually not put the entire lorax bin/ directory into your path, as this may cause binaries from the virtual environment such as python to be used in contexts where they were not intended to be used. The easiest way to do this is with a symlink to a directory on your path, such as the Anaconda Python directory:

```
ln -s /path/to/anaconda/envs/loraxenv/bin/lorax_env /path/to/anaconda/bin
```

Deactivate virtual environment

You no longer need to work inside the virtual environment you created. Leave the virtual environment with the following command:

```
source deactivate
```

Next, follow the configuration instructions.

Testing

If you defined the lorax host to be something other than localhost, you will have to define the environmental variable LORAX_HOST to match:

```
export LORAX_HOST=MY_HOST_IP
```

Ditto for the port address.

To test lorax, issue the commands:

```
mkdir test_lorax # could be any name, will be deleted later
cd test_lorax
/path/to/lorax_env lorax create_test_files
./test_targets.sh
```

If the installation went properly, the last command should finish with "lorax test completed successfully."

You may also wish to load a full-sized data model to work with. The script `get_big_model.sh` will do that for you. The model of 12 legume genomes (against the phytozome 10.2 HMM's) is rather large at 900 MB of downloads and some 7 GB of data when loaded, so expect this to take a while (about 30 minutes on a core i7 with SSD). This script will define some 18,000 legume gene families.

If you wish to stop lorax after the tests, issue the command:

```
/path/to/lorax_env supervisorctl shutdown
```

Configuration

The design of `lorax` is necessarily complex, with configuration occurring in multiple layers at different times. Proceeding in order from highest to lowest priority, these are:

Configuration Variables

Manual Configuration

Configuring lorax uses the `lorax config` command, run in the proper environment using the lorax_env script. The path to this script is the value of LORAX_ROOT you chose at install time:

```
/path/to/lorax_env -i
```

You will now get a `lorax_env>` prompt and you are ready to configure. For example:

```
lorax config host MY_IP_ADDRESS
lorax config crashmail_email MY_EMAIL_ADDRESS
lorax config secret_key # prints value of secret_key
lorax create_instance
```

Now exit the lorax_env shell with control-D. Run lorax and its associated processes:

```
/path/to/lorax_env -v start
```

The start command, when run with the `-v` switch should return a list of started processes, all with status RUNNING.

Configuring by Script

If you did a direct installation using the `lorax_tool` script, you should review and edit the `my_config.sh` script to reflect the settings you wish to use use for your installation. Then run `./lorax_tool configure_pkg` to do the build.

Running lorax

`lorax` can be run via command line with no arguments.

If asynchronous queues are switched on (modes other than `development`), then `redis` will need to be started at the address specified by the configuration variable `RQ_REDIS_URL`. After that, two RQ work queues will need to be started as well:

```
rqworker treebuilding
rqworker alignment
```

It is also useful to run a dashboard (via `rq-dashboard`).

`lorax` is intended to be run in a trusted environment and contains no authentication. It should be run on ports that are accessible only to trusted hosts. Running `lorax` on a public port opens the possibility of denial-of-service attacks.

We recommend that `lorax` be run in a virtual environment if on a shared server. However, the shell scripts will work for real environments as well.

URLs

`lorax` interacts through standard http `GET` and `POST` commands. POST of sequence data and calculation of trees results in permanent URLs that `lorax` will serve up until such time as the corresponding disk entries are deleted. POSTing to existing sequence data will result in over-writing.

URL	Interpretation
/log.txt	A GET of this URL returns the log file of the current run.
/rq	A dashboard of the queues used by RQ, if asynchronous queueing is in use.
/trees/families.json	A GET of this URL returns a JSON list of defined families.
/trees/<family>/sequences	POST a FASTA-formatted set of aligned sequences. ID fields must be unique and use UTF-8 encoding. If the multipart key is peptide, the sequences are assumed to be protein sequences; if the key is DNA, DNA sequences are assumed. <family> is used as a directory name and must not include special characters such as /. See the ''post_to_lorax.sh'' script in the test/ directory for an example of how to post. Returns a JSON object that gives information about the number and length of submitted sequences. Throws a 400 error if a proper multipart key is not found. Throws a 406 error if empty or improperly-formatted FASTA file is sent.
/trees/<family>/alignment	POST a FASTA-formatted set of aligned sequences. Same as for sequences above, except dash (-) characters are used as spacings in alignments.
/trees/<family>/HMM	PUT a family HMM for use with hmmalign. Throws a 400 error if family has not been previously created. Returns a JSON dictionary of HMM stats. Throws a 406 error if the HMM definition is not valid.
/trees/<family>/hmmalign	A GET of this URL will cause an HMM alignment to be calculated. This step is not needed if an alignment is supplied. Throws a 400 error if a sequence file is not found. Throws a 417 error if hmmalign returned a non-zero process code.
/trees/<family>/FastTree	A GET of this URL will cause a FastTree tree to be calculated and a Newick tree to be returned. This operation may take a long time and result in a timeout, which is why polling methods are provided. Throws a 417 error if an improper family name was provided. Throws a 404 error if family was not previously created. Throws a 405 error if an alignment is not found. Throws a 417 error if
/trees/<family>/RAxML	Same as above, execpt using RAxML as the tree builder.
/trees/<fam>/hmmalign_FastTree	A GET of this URL will cause the alignment and tree- building steps to be chained.
/trees/<fam>/<meth>/status	Returns -1 if tree calculation is ongoing, and the exit code of the tree-builder <meth> if calculation is complete.
/trees/<fam>/<meth>/tree.nwk	Returns already-calculated tree in Newick format.
/trees/<fam>/<meth>/tree.xml	Returns already-calculated tree in phyloXML format.
/trees/<fam>/<meth>/run_log.txt	Returns the log file, including timings, of the tree calculation.
/trees/<fam>.<super>/sequences	POST additional sequences to be considered a superfamily of existing family <fam>. <super> cannot be a reserved name such as FastTree. These sequences will be concatenated to the existing family sequences, with <super> prepending ID strings.
/trees/<family>.<superfamily>/	DELETE a superfamily.
/trees/<f>.<s>/<meth>/status	Returns status of a superfamily tree calculation.
/trees/<f>.<s>/<meth>/tree.nwk	Returns tree of a superfamily in Newick format.
/trees/<f>.<s>/<meth>/tree.xml	Returns tree of a superfamily in phyloXML format.
/trees/<f>.<s>/<m>/run_log.txt	Returns the log file of a superfamily tree calculation.

Monitoring

There is monitoring of `lorax` and associated processes via web interfaces at multiple levels:

Address	Type of monitor
SUPERVISOR_HOST:SUPERVISOR_PORT	Supervisord status and controls (password-protected).
LORAX_HOST:LORAX_PORT/rq	RQ queue info, including errors.

The default supervisord port is 58928.

lorax package

Submodules

lorax.cli module

lorax.config module

lorax.config_file module

lorax.filesystem module

lorax.logging module

lorax.version module

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search