NPSGD User Guide

Thomas Dimson
Natural Phenomena Simulation Group (NPSG)
David R. Cheriton School of Computer Science
University of Waterloo, Canada

January 30, 2011

Abstract

NPSGD is an online framework that makes it easy for researchers to exhibit their models online. This document provides a general overview of how the system works along with some administration pointers. Notably, it does not address the inner workings of how to create models as this is documented in the guide called "Adding a new model to NPSGD".

Contents

1 System Requirements

A basic configuration will require:

- Python 2.5 or higher (`http://www.python.org/`)
- The Tornado Web server for Python (`http://www.tornadoweb.org/`)
- LaTeX distribution of some form.
- UNIX-like operating system. NPSGD has been tested on Ubuntu Linux 9.04 and 10.04 but should work on other Linux or BSD based platforms.

The software has very low hardware requirements. It has been stress tested on a single CPU virtual machine with 384 megabytes of RAM.

2 System Overview

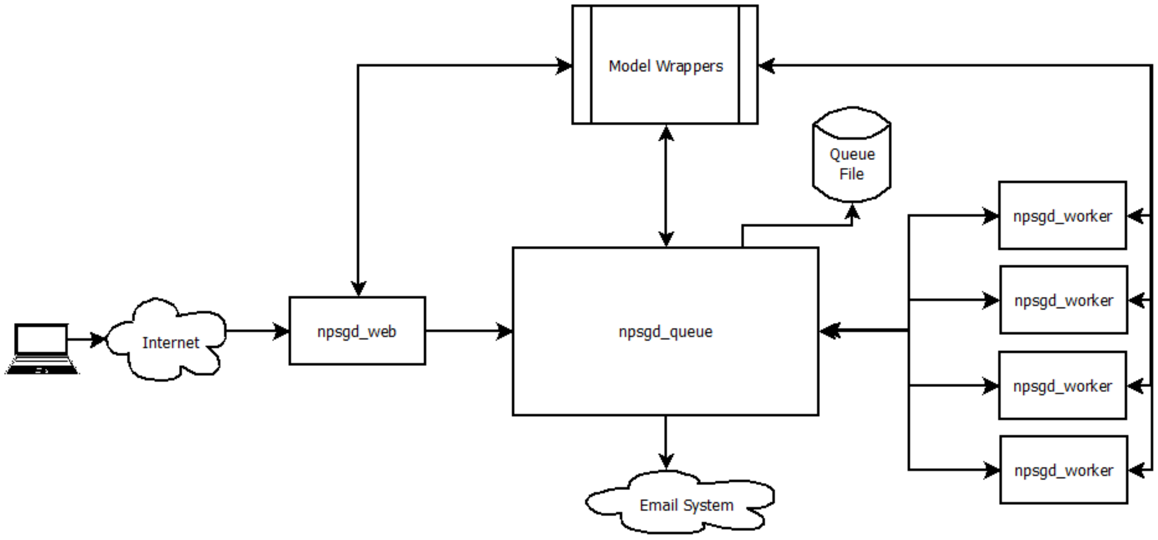

Figure 1: Information Flow in NPSGD

Figure 1 shows a high level overview of the information flow within NPSGD. This shows the only client facing daemon, `npsgd_web` serving requests to the user directly. All daemons have a dependency on the model wrappers and thus the source must be synced between them. As a request is populated, it is flowed to a disk-backed queue which is polled periodically by the workers. Once a worker completes a request, it will notify the user via e-mail.

2.1 Daemons

NPSGD is a distributed system which communicates internally and externally via HTTP. One of the design goals of the system was to allow multiple machines to perform as workers to avoid to overloading any particular system. To facilitate this, the framework is split into three daemons:

- `npsgd_web`: The frontend to NPSGD. The job of this daemon is to communicate with all external clients over HTTP. Since no state is kept inside the daemon, there is no limit to the number of frontends that exist (though one should be sufficient unless you are under heavy load). By default, a client visiting `http://localhost:8000` will hit this daemon.
- `npsgd_queue`: The manager of NPSGD. The job of this daemon is to keep track of the state of particular model runs, send out e-mail confirmations, and to distribute jobs to worker machines. There can only ever be one queue within an NPSGD system. By default, it runs on port 9000 but should not be accessed externally (in particular, it should be firewalled off). The queue is a listener for other requests with the `npsgd_web` and `npsgd_queue` daemons communicating directly with the queue via HTTP over port 9000.
- `npsgd_worker`: The model processor of NPSGD. The job of these daemons are to poll the queue for model requests and process them. Generally, this takes the form of a finite state machine that spawns some external process to actually perform the scientific simulation. After processing, it sends results to the requestor's e-mail address.

If NPSGD is being configured for long term access, all of these daemons should have startup scripts associated on the end machine. Sample scripts are available in the `startup/` directory alongside the NPSGD distribution.

Internal communication between daemons is performed over HTTP. Parameters are passed via URL-encoded JSON that is present in the body of requests for POST requests, and as parameters for GET requests. For detailed information on the structure of internal requests, please see their implementations in `npsgd_web.py`, `npsgd_queue.py` and `npsgd_worker.py`. Additionally, there is a "secret" that is passed in plaintext along every HTTP request communicating with the queue. A secret is a string, specified in the config file, that must match on both client and queue. This is **not** a substitute for a firewall. The queue should be configured to only accept traffic on port 9000 from known hosts (those that run `npsgd_web` and `npsgd_worker`).

Each model that is implemented for NPSGD must be available for the queue, the web frontend and the worker machines. A NFS mount for the models is particularly handy for this purpose (though not necessary).

2.2 Request Flow

A sample model of NPSGD is available at `http://www.npsg.uwaterloo.ca/models/ABMU.php`. It would be helpful to try this out in order to gain understanding of the system. The typical request flow would be something like this:

1. User visits the model site via `http://localhost:8000/models/example`, which is served using `npsgd_web`.
2. User submits a request for the example model, `npsgd_web` acquires a confirmation code from `npsgd_queue`.
3. `npsgd_queue` e-mails the same confirmation code to the user's e-mail address.
4. User clicks the link in the e-mail, usually something like `http://localhost:8000/confirm_submission/code`.
5. `npsgd_web` forwards the confirmation onto `npsgd_queue`. At this point the model request is ready to be processed.
6. `npsgd_worker` has been polling the queue for a task. Since now there is a task, it receives a task response.
7. `npsgd_worker` prepares execution, spawns a subprocess to run the underlying model.
8. Model execution completes. `npsgd_worker` creates a LaTeX document with the results.
9. LaTeX document and other attachments are sent to the user's e-mail. `npsgd_worker` tells the `npsgd_queue` that it has executed successfully.

2.3 Fault Tolerance

NPSGD has been designed to be fault tolerant. Whenever a web request is made, `npsgd_web` will check with the queue to ensure it is up and that it has workers on the line that have polled for a request recently. If one of these conditions is not met, a simple error message is displayed to the requester.

When processing a task, `npsgd_workers` communicate back to the queue with a keepalive request to make sure that the queue knows that the worker is still alive. If this keepalive has not been heard for a long while, the queue daemon will put the request back at the tail of the queue and increase its failure count.

All requests are tried for a fixed number of times before a complete failure (by default, the queue will retry a request three times before declaring it failed). Upon failure, an error message is e-mailed to the user.

If the e-mail system is ever down, `npsgd_queue` will keep trying to send e-mails out until they have successfully completed.

`npsgd_queue` can be shut down with no data loss (even if requests are in the queue). The queue server is serialized to disk (using Python's `shelve` standard module) every time a new request jumps into the queue or one is taken out. When the queue boots up, it loads back all the requests that had confirmation codes or had model requests in the queue. If a model has been updated since the queue went down, there will be no way to complete the request so an e-mail is sent to the requester to notify them of failure. It is likely that they can simply retry their request at that point.

3 Software Layout

NPSGD ships in exactly the same way it was developed and is not easily converted into a form that matches the standard UNIX layouts. When you clone the main repository for NPSGD, you will see the following files and directories:

- `config.example`: An example configuration file for NPSGD.
- `epydoc.cfg`: Config file for generating an HTML representation of the code using epydoc.
- `LICENSE`: License agreement for distribution.
- `README`: Easy readme guide to NPSGD.
- `npsgd\_web.py`: Web daemon for handling client requests.
- `npsgd\_queue.py`: Daemon for queueing NPSGD model runs.
- `npsgd\_worker.py`: Worker for actually processing NPSGD model runs.
- `doc/`: Directory containing NPSGD documentation.
- `models/`: Default directory for storing user-defined models.
- `npsgd/`: Helper package containing modules required to run the daemons.
- `startup/`: Directory containing some sample startup scripts for Ubuntu (must be modified to specify paths correctly).
- `static/`: Directory containing static files served by the web server (Javascript, css).
- `templates/`: Directory containing templates used by NPSGD.

3.1 Templates

Templates are used in many different ways within NPSGD:

- To create the HTML that is served by the web daemon.
- To create the e-mail sent by the queue and worker daemons.
- To create LaTeX markup to send results to the user.

All templates for NPSGD are stored in the `templates/` subdirectory. This is meant to be user defined. The default templates shipped with NPGSD are specific to the models within the Natural Phenomena Simulation Group and are intended to be used as a guide for other NPSGD users.

Templates use a syntax created for the Tornado web server. This syntax is documented at the Tornado site: `http://www.tornadoweb.org/`, under templates.

3.1.1 HTML Templates

NPSGD ships with two sets of HTML templates, one for an embedded site (served directly without entities like title tags), and one for a basic site. These templates are used by the web daemon to display HTML directly to the user. Both are available under the `templates/html/` subdirectory. These files contain:

1. `base.html`: Template that all other templates inherit from. It could contain entities like HTML headers and footers.
2. `model.html`: The template that displays a model to the user.
3. `model_error.html`: The template that displays a message when an error occurs in the model.
4. `confirm.html`: Template that displays after a model request has been made (displays that an e-mail will be dispatched to the user).
5. `confirmed.html`: Template that displays after a confirmation code has been entered correctly.
6. `already_confirmed.html`: Template that displays after a confirmation code has been entered for a second time (displays a message saying that the model will not be rerun).

3.1.2 Email Templates

Emails are used as communication to the user after a request has been performed via HTML. Each template comes in two parts: one for the email subject, and one for the email body. The naming convention is "name_body.txt" for body and "name_subject.txt" for subject. In particular, these templates are available:

- `results_email`: The email used to display model run results.
- `failure_email`: The email used to notify the user when a certain error threshold has been met during across model execution attempts.
- `confirm_email`: The email used to communicate the confirmation code for a particular model request.
- `confirmation_failed_email`: The email that is used to communicate that a confirmation code has expired prematurely. This is only sent if the user receives a confirmation code, the queue goes down, then comes up with different model versions.
- `lost_task_email`: The email that is used to communicate that we are no longer able to execute a particular model request. This is sent only if we are processing a model request, the queue goes down and then comes up with different model versions.

3.1.3 LaTeX Templates

There is only one LaTeX template, namely `result\_template.tex`. This is used to declare all the packages required for a model results PDF. Inside, there is an empty variable where the details of a model run will go. Models themselves specify the model details, but it is always wrapped in this template.

4 Development Environment

Since NPSGD consists of so many components, creating a quick development environment may seem overwhelming. First of all, ensure that all the requirements mentioned in Section 1 are installed. After that, clone NPSGD from the main development site and create a custom copy of the config file:

```
git clone git://github.com/cosbynator/NPSGD.git
cp config.example config.cfg
```

The config file will need a few changes to run, notably:

- Change `npsgbase` to point at the directory that you cloned the software into (e.g., `/home/tdimson/npsgd`).
- Change `pdflatexPath` to point at your `pdflatex` binary.
- Change the `email` section to specify a valid username/password and to enter your SMTP server. E-mail uses the SMTP protocol and has configuration options for toggling various SMTP features (notably encryption and SMTPAUTH). The defaults are set up for a gmail account (provided you specify a valid username/password). Any provider supporting SMTP should be compatible with NPSGD.
- Change `matlabPath` if you will be testing/editing Matlab models.

After completing these sections you will be able to run all three daemons on a local terminal. Specifically, run the following in separate terminal windows:

```
python npsgd_web.py
python npsgd_queue.py
python npsgd_worker.py
```

Logging will be printed to standard error unless otherwise specified (using the `-l` command line option). After running `npsgd_web` you should be able to browse to `http://localhost:8000/models/example` and see some sample output. Try performing a complete model run to make sure the system is operating correctly.

5 A word on models

Models are the key component to customizing NPSGD. The scope of this document does not cover the *implementation* of custom models (see "Adding a new model to NPSGD" from `http://www.npsg.uwaterloo.ca` for that). Still, it is important to understand how they fit into the system.

Models are Python files that inherit from `ModelTask` in order to provide complete flexibility for underlying implementation. These Python files often act as wrappers to existing models that are implemented in some other language (we have models implemented in both Matlab and C++). This does not force the model implementation to be in Python; Python acts as a facilitator between NPSGD and the underyling model.

Each daemon is configured to monitor the model directory, by default under `models/`. Python files in this directory act as "plugins" to the system - models are implemented in them. The daemons periodically scan the directory and look at the all the python files, those that inherit from `ModelTask` are imported into the system. When importing, a hash of the file is used to tag the model with a "version". Duplicate versions of models are ignored (not imported into the system), while new versions are stored. Old versions are retained in the system in case there are any queued requests that are still acting on the old version. For this reason, it is important for the developer to reload the page after updating any models if they wish to test the new version.

Models are self-contained: they have all the information necessary for the queue, web and worker daemons to operate. In particular, they contain the parameters necessary for the web interface to configure the models and the execution cycle necessary in order for the workers to run a model with a particular parameter set. The code for the models must be shared across the daemons: this is done manually, or via a NFS mount.

6 Static Files

In addition to templates, NPSGD needs the use of certain static files to customize the in-browser behaviour. All static files are available in the `static/` subdirectory.

6.1 Javascript

NPSGD uses Javascript to handle parameter verification and provide UI widgets for the user. The following are located in the `static/js` subdirectory:

1. `npsgd.js`: Our particular javascript files.
2. `jquery-version.min.js`: JQuery, a javascript library that makes writing javascript simpler: `http://jquery.com/`
3. `jquery.qtip.min.js`: A library for creating popup tooltips. This is used for model's helper text. `http://craigsworks.com/projects/qtip/`
4. `jquery-ui-version.min.js`: JQuery UI, a set of UI widgets for JQuery: `http://jqueryui.com/`. This is used to provide an interface for widgets such as sliders and range selectors.
5. `jquery.validate.min.js`: JQuery validation plugin. This is used to perform parameter verification in client side, before requests are sent to the server: `http://bassistance.de/jquery-plugins/jquery-plugin-validation/`

6.2 CSS

In addition to Javascript, some basic CSS needs to be shared across all templates. These are in the `static/css` subdirectory.

1. `npsgd.css`: This provides just a couple of CSS includes that are needed across all templates.
2. `smoothness/`: This subdiretory contains the default CSS files for the JQuery UI project. It is used to dislay the widgets of JQuery UI.

6.3 Images

Finally, there are some images that NPSGD needs to display available in the `static/images` subdirectory.