Sheepdog Documentation

Release 0.1.1

Adam Greig

March 21, 2014

Sheepdog effects GridEngine jobs without affecting your affect.

Contents:

CHAPTER 1

Quickstart

Woof woof!

1.1 Requirements

1.1.1 A GridEngine Cluster

Your cluster must have some *head* node, which is the node you connect to when you want to run `qsub`. Here we'll use the name `fear`, because that is the name of the author's head node.

The head node must be able to run `qsub` and you must be able to SSH into it. This might require having GridEngine stuff in your `.bashrc`, so that `ssh fear qstat -F no` actually works.

The cluster workers must be able to run a Python interpreter (you can specify the path if you wish to use a custom interpreter).

The cluster workers must be able to connect to the computer running Sheepdog on a TCP port (default 7676 but may be specified).

Your GridEngine must support array jobs (the -t command).

1.1.2 The `fear` GridEngine Cluster

If you're also using `fear`, put this in your `.bashrc`:

```
source /usr/local/grid/divf2/common/settings.sh
```

1.1.3 Local Python

Locally you must have Flask and Paramiko installed. If you have Tornado installed it will be used instead of the Flask debug server, as it is faster and better. To run tests Nose is required.

1.2 Synchronous Map

In the simplest case you have some function `f(x1, x2, ...)` and you wish to run it with many arguments, `[(a1, a2, ...), (b1, b2, ...), ...]` and get back the results, `[f(a1, a2, ...), f(b1,`

`b2, ...), ...]`. If the results are likely to come in quickly and/or you just want to wait for them, use `sheepdog.map_sync()`.

Here's what it looks like:

```
>>> import sheepdog
>>> def f(a, b):
...     return a + b
...
>>> args = [(1, 1), (1, 2), (2, 2)]
>>> conf = {"host": "fear"}
>>> sheepdog.map_sync(f, args, conf)
[2, 3, 4]
```

1.3 Namespaces

Often the target function will require other items be present in its namespace, for instance constants or other functions. These may be passed in the namespace parameter *ns* of map_sync:

```
>>> import sheepdog
>>> constant = 12
>>> def g(x):
...     return x * 2
...
>>> def f(a, b):
...     return a + g(b) + constant
...
>>> args = [(1, 2), (2, 3), (3, 4)]
>>> conf = {"host": "fear"}
>>> namespace = {"constant": constant, "g": g}
>>> sheepdog.map_sync(f, args, conf)
[17, 20, 23]
```

1.4 Imports

Sheepdog doesn't currently provide for automatic handling of imports and dependencies. Please ensure that all required Python packages are available on the execution hosts. To actually run the import, put it at the top of your function, optionally exporting the package so that other functions can use it.

For example:

```
>>> def g(x):
...     return np.mean(x)
...
>>> def f(x):
...     import numpy as np
...     global np
...     return g(x)
...
```

CHAPTER 2

The Configuration Object

The `config` dictionary passed to the top level functions controls how Sheepdog behaves. These are the available options:

2.1 SSH Options

2.1.1 `host` (required)

The hostname to submit GridEngine jobs to. This is the server you normally SSH into to run `qsub` etc.

This must be specified and has no default value.

2.1.2 `ssh_port`

The SSH port to connect to. Defaults to 22.

2.1.3 `ssh_user`

The username to connect with. Defaults to the current system user.

2.1.4 `ssh_dir`

The remote directory to place job scripts in. Relative paths will be relative to the user's home directory. Defaults to `.sheepdog`.

2.2 Local Server Options

2.2.1 `dbfile`

The file (or path) to store the sqlite database in. Since results are kept between requests in case you want to get them later, it might be nice to have database per set of related projects. Or per project. Or per request, whatever.

Defaults to `./sheepdog.sqlite`.

2.2.2 port

The port that the local HTTP server will listen on. The GridEngine clients must be able to connect to the local computer on this port.

Defaults to 7676.

2.2.3 localhost

The hostname by which GridEngine workers may contact the local server. Defaults to the local FQDN (which really should work!)

2.3 GridEngine Options

2.3.1 shell

A string containing the Python interpreter to use to execute the script. This is passed to the GridEngine -S option and placed on the script shebang.

Should be a Python binary which the GridEngine worker can execute.

Defaults to `/usr/bin/python`.

2.3.2 ge_opts

A list of strings containing GridEngine options. This is used to specify additional GridEngine related arguments, for example `-l ubuntu=1` to specify a resource requirement or `-r y` to specify that the job may be re-run.

If unspecified, the defaults are:

```
["-r y", "-l lr=0", "-l ubuntu=1",
 "-wd $HOME/.sheepdog/", "-o $HOME/.sheepdog/", "-e $HOME/.sheepdog/"]
```

which is particularly helpful on `fear` and shouldn't be too adverse elsewhere.

Note that `-S /path/to/shell` is always specified by the `shell` option detailed above, and `-t 1-N` is always specified with N equal to the number of arguments being evaluated.

All these options are written to the top of the job file which is copied to the GridEngine server, so may be inspected manually too.

CHAPTER 3

Changelog

3.1 Version 0.1

3.1.1 0.1.8

Released on 2014-03-21.

- Swap to Paramiko for SSH usage. Much nicer.
- Swap to urllib rather than Requests. A pity, but removes the dependency.
- Fix Tornado starting from inside IPython Notebook.
- Clients now print out their results so GridEngine can save it in the .o files

3.1.2 0.1.7

Released on 2014-03-21.

- Fix Py2 by using list() instead of list.copy()

3.1.3 0.1.6

Released on 2014-03-20.

- Fix tests for namespace serialisation.

3.1.4 0.1.5

Released on 2014-03-20.

- Fix bug where ge_opts would be appended to every map_sync call
- Fix bug where functions in the request namespace only got a copy of the namespace so global imports etc would not work

3.1.5 0.1.4

Released on 2014-03-20.

- Improve test coverage
- Refactor all default values to sheepdog/__init__.py
- **Improved defaults:**
  - Use ~/.sheepdog as the default working directory on the remote host
  - Use /usr/bin/python instead of /usr/bin/env python as this confuses GE
  - Quote user-provided shells in case they contain a space

3.1.6 0.1.3

Released on 2014-01-21.

- Change package layout to remove subpackages, because flat is better.
- Improve docstrings.
- Refactor serialisation to its own module which is used throughout Sheepdog.
- Store job files in ~/.sheepdog on remote server

3.1.7 0.1.2

Released on 2013-12-05.

- Adds the Requests package to requirements as you can't actually run the local code without it.

3.1.8 0.1.1

Released on 2013-12-04.

- Adds Python 2.7 compatibility by frobbing some bytes() in the sqlite stuff.

3.1.9 0.1.0

Released on 2013-12-04. First release.

- Contains `sheepdog.map_sync()`, the first top level utility function, plus the basic underlying sqlite storage and tornado/flask web server bits.

CHAPTER 4

sheepdog

4.1 sheepdog Package

4.1.1 sheepdog Package

Sheepdog is a utility to run arbitary code on a GridEngine cluster and collect the results, typically by mapping a set of arguments to one function.

Documentation: http://sheepdog.readthedocs.org

Source code: https://github.com/adamgreig/sheepdog

PyPI: https://pypi.python.org/pypi/Sheepdog

Sheepdog is released under the MIT license, see the LICENSE file for details.

sheepdog.__init__.**map_sync**(*f*, *args*, *config*, *ns=None*)

Run *f* with each of *args* on GridEngine and return the results.

Optionally *ns* is a dict containing a namespace to execute the function in, which may itself contain additional functions.

Blocks until all results are in.

config must be a dict including:

host: the hostname to submit grid engine jobs to [required]

***ssh_port*: the ssh port to connect on** (default: 22)

***ssh_user*: the ssh username to use** (default: current username)

***ssh_dir*: the remote directory to put job scripts in,** relative to home directory if a relative path is given (default .sheepdog)

***dbfile*: the filename for the results db** (default ./sheepdog.sqlite)

***port*: the port for the server to listen on** (default: 7676)

***ge_opts*: a list of grid engine options**

(default: ['-r y', '-l ubuntu=1', '-l lr=0', "-wd $HOME/.sheepdog/", "-o $HOME/.sheepdog/", "-e $HOME/.sheepdog/"])

***shell*: the path to the python to run the job with** (default: "/usr/bin/python")

***localhost*: the hostname for workers to find the local host** (default: system's FQDN)

4.1.2 client Module

Sheepdog's clientside code.

This code is typically only run on the worker, and this file is currently only used by pasting it into a job file (as workers don't generally have sheepdog itself installed).

class sheepdog.client.**Client**(*url*, *request_id*, *job_index*)
Find out what to do, do it, report back.

HTTP_RETRIES = 10

get_details()
Retrieve the function to run and arguments to run with from the server.

go()
Call get_details(), run(), submit_results(). Just for convenience.

run()
Run the downloaded function, storing the result.

submit_results()

4.1.3 deployment Module

Code for deploying code to servers and executing jobs on GridEngine.

class sheepdog.deployment.**Deployer**(*host*, *port*, *user*)
Connect to a remote SSH server, copy a file over, run qsub.

__init__ takes (host, port, user) to specify which SSH server to connect to and how to connect to it.

deploy(*jobfile*, *request_id*, *directory*)
Copy *jobfile* (a string of the file contents) to the connected remote host, placing it in *directory* with a filename containing *request_id*.

submit(*request_id*, *directory*)
Submit a job to the GridEngine cluster on the connected remote host. Calls qsub with the job identified by request_id and directory.

4.1.4 job_file Module

Generate job files to send to the cluster.

The template is filled in with the job specifics and the formatted string is returned ready for deployment.

sheepdog.job_file.**job_file**(*url*, *request_id*, *n_args*, *shell*, *grid_engine_opts*)
Format the template for a specific job, ready for deployment.

url is the URL (including port) that the workers should contact to fetch job information, including a trailing slash.

request_id is the request ID workers should use to associate themselves with the correct request.

n_args is the number of jobs that will be queued in the array task, the same as the number of arguments being mapped by sheepdog.

shell is the path to the Python that will execute the job. Could be a system or user Python, so long as it meets the Sheepdog requirements. Is used for the -S option to GridEngine as well as the script shebang.

grid_engine_opts is a list of string arguments to Grid Engine to specify options such as resource requirements.

4.1.5 server Module

Sheepdog's HTTP server endpoints.

The Server class sets up a server on another subprocess, ready to receive requests from workers. Uses Tornado if available, else falls back to the Flask debug web server.

class sheepdog.server.**Server**(*port=7676*, *dbfile=None*)
Run the HTTP server for workers to request arguments and return results.

__init__ creates and starts the HTTP server.

stop()
Terminate the HTTP server.

sheepdog.server.**get_config**()
Endpoint for workers to fetch their configuration before execution. Workers should specify *request_id* (integer) and *job_index* (integer) from their job file.

Returns a JSON object:

{"func": (serialised function object), "args": (serialised arguments list)

}

with HTTP status 200 on success.

sheepdog.server.**get_storage**()
Retrieve the request-local database connection, creating it if required.

sheepdog.server.**report_error**()
Endpoint for workers to report back errors in function execution. Workers should specify *request_id* (integer), *job_index* (integer) and *error* (an error string) HTTP POST parameters.

Returns the string "OK" and HTTP 200 on success.

sheepdog.server.**run_server**(*port=7676*, *dbfile=None*)
Start up the HTTP server. If Tornado is available it will be used, else fall back to the Flask debug server.

sheepdog.server.**submit_result**()
Endpoint for workers to submit results arising from successful function execution. Should specify *request_id* (integer), *job_index* (integer) and *result* (serialised result) HTTP POST parameters.

Returns the string "OK" and HTTP 200 on success.

4.1.6 storage Module

Interface to the storage backend.

Future plans involve porting most of those handwritten SQL to a sensible ORM.

class sheepdog.storage.**Storage**(*dbfile='./sheepdog.sqlite'*)
Manage persistence for requests and results.

Request functions and result objects are stored as binary blobs in the database, so any bytes object will be fine. They'll be returned as they were sent.

__init__ creates a database connection.

dbfile is a file path for the sqlite file, or *:memory:* to only use in memory persistence.

count_errors(*request_id*)
Count the number of errors reported so far.

count_results(*request_id*)
Count the number of results so far for the given request_id.

count_results_and_errors(*request_id*)
Sum the result and error counts.

count_tasks(*request_id*)
Count the total number of tasks for this request.

get_details(*request_id*, *job_index*)
Get the target function, namespace and arguments for a given job.

get_errors(*request_id*)
Fetch all errors for a given request_id.

Returns a list of (args, error) items in the order of the original args_list provided to new_request.

get_results(*request_id*)
Fetch all results for a given request_id.

Returns a list of (args, result) items in the order of the original args_list provided to new_request.

Gaps are not filled in, so if results have not yet been submitted the corresponding arguments will not appear in this list and this list will be shorter than the length of args_list.

initdb()
Create the database structure if it doesn't already exist.

new_request(*serialised_function*, *serialised_namespace*, *args_list*)
Add a new request to the database.

serialised_function is some bytes object that should be given to workers to turn into the code to execute.

serialised_namespace is some bytes object that should be given to workers alongside the serialised function to provide helper variables and functions that the primary function requires.

args_list is a list, tuple or other iterable where each item is some bytes object that should be given to workers to run their target function with.

Returns the new request ID.

store_error(*request_id*, *job_index*, *error*)
Store an error resulting from a computation.

store_result(*request_id*, *job_index*, *result*)
Store a new result from a given request_id and job_index.

CHAPTER 5

Indices and tables

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

S