Sheepdog Documentation

Release 0.1.1

Adam Greig

January 12, 2015

Contents

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 (defaults to a random available port 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()`.

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", "ge_opts": ["-l ubuntu=1", "-l lr=0"]}
>>> sheepdog.map(f, args, conf)
[2, 3, 4]
```

1.3 Asynchronous Map

Much like `sheepdog.map()`, `sheepdog.map_async()` runs `f` with each set of arguments in `args` using the provided configuration and optional namespace. Unlike `sheepdog.map()`, `sheepdog.map_async()` returns a request ID immediately after deployment, and it is then up to the user to poll for status, for example using `sheepdog.get_results()`.

1.4 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:

```
>>> 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]
```

1.5 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)
...
```

1.6 Results and Errors

To fetch results out of the database after a request, see `sheepdog.get_results()`. Similarly for errors that may have occured during the job (those that Python was able to catch and recover from), you can use `sheepdog.get_errors()`. If errors were detected and verbose mode is on, you will also be prompted to check the errors after calling `sheepdog.map()`.

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_keyfile`

The path to the SSH key that should be used. The key must be passphraseless. Defaults to seeing if any key from an SSH agent or *.ssh/id_rsa* or *.ssh/id_dsa* may be used. Use of an SSH agent is recommended.

2.1.5 `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 None, which will cause Sheepdog to find an available high-numbered port and use that. Specify a particular port number if you wish to run on a specific port.

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:

```
["-wd $HOME/.sheepdog/", "-o $HOME/.sheepdog/", "-e $HOME/.sheepdog/"]
```

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.

If the resource specification `-l mem_grab=2G` (2G for example) is present, the sheepdog client will automatically call `resource.setrlimit` to restrict the process to that amount of memory.

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

SSH

Sheepdog uses SSH to connect to the GridEngine cluster head. There are a few issues that may come up in the process.

3.1 SSH Keys

Sheepdog will only use SSH keys to connect to the remote server. If you don't already have these set up, it is simple to do so:

```
local$ ssh-keygen
local$ ssh-copy-id remote
```

Additionally the keys should either be passphraseless (inadvisable) or stored in an SSH agent, which Sheepdog will use automatically. Most operating systems will automatically set up an SSH agent for you, and you can either connect to the host manually to add the key to the agent, or use `ssh-add`.

Sheepdog will automatically find a key named *id_rsa* or *id_rsa* in *~/.ssh*, or you can set *ssh_keyfile* to a path to the (passphraseless) key file to use. The best way is still to use an SSH agent, though!

3.2 Known Hosts

Sheepdog uses Paramiko to connect to SSH servers, and instructs it to read your ~/.ssh/known_hosts file to collect information on host keys. It does not permit it to connect without a valid known host key.

However this can cause issues on remote hosts which use ECDSA keys and also offer an RSA key. In this case Paramiko will (at time of writing) request the RSA key, fail to find it in your known_hosts (which is likely to only contain the ECDSA key) and refuse to connect to the server.

One workaround for this problem is to fetch the RSA key of the server and place it into your known_hosts, for instance:

```
local$ ssh remote ssh-keyscan -t rsa remote >> ~/.ssh/known_hosts
local$ ssh-keygen -H
```

3.3 SSH Config

If a file `~/.ssh/config` exists, Sheepdog will use Paramiko to read this file and use it to determine hostnames, usernames and ports to connect to. In addition, ProxyCommand directives will also be followed. No other configuration parameters are used.

If a hostname, username or port is found in the SSH config that matches the provided hostname, they will be used in preference to the `ssh_user` and `ssh_port` configuration options.

CHAPTER 4

Tips and Tricks

4.1 Setting Custom Pickle and Marshal Protocols

By default, Sheepdog uses Pickle protocol *pickle.DEFAULT_PROTOCOL* and marshal version *marshal.version*. These correlate to the default protocols for your current Python interpreter.

However, if your worker nodes are running an older Python than the computer running Sheepdog, you may need to decrease these protocols. For example, Python 3 uses Pickle protocol 3 by default, which is not compatible with any Python 2.

To change this, overwrite *sheepdog.serialisation.pickle_protocol* and *sheepdog.serialisation.marshal_version*:

```
>>> import sheepdog
>>> sheepdog.serialisation.pickle_protocol = 2
>>> sheepdog.map_sync(...)
```

4.2 Restarting The Server

Should the HTTP server that listens for requests and results from job workers die, any still-queued jobs will not be able to start, and any currently running jobs will not be able to submit results. Jobs do send their results to standard output, but it's obviously better to get a server up again. You don't have to resubmit the request, instead just start a new Server and keep it around until the job is complete. Here's an example:

```
import sys
import time
import sheepdog
server = sheepdog.Server()
storage = sheepdog.Storage()
while True:
    n_results = storage.count_results(int(sys.argv[1]))
    print(n_results, "results\r", flush=True)
    time.sleep(5)
```

This code gets one request ID from the command line, starts a server (here using the default location of './sheepdog.sqlite' but you can change this in the Storage constructor) and waits for results to come in.

4.3 Finding Request IDs, Ports and Passwords

For getting results out of previously completed jobs you'll need the request ID, but this isn't obviously found if you're just using the synchronous `sheepdog.map()` function. You can find these values by looking in the Sheepdog job files on the GridEngine server, instead. These files are by default placed in *~/.sheepdog* and named like *sheepdog_032.py*. The final line contains a call to Client with the URL, password and request ID:

```
Client("http://your.host:1234", "hunter2", 12, job_index).go()
```

In this example the request ID is 12.

CHAPTER 5

Changelog

5.1 Version 0.2

Starting to become stable and useful.

5.1.1 0.2.3

Released on 2015-01-12

- Use a 30 second timeout on sqlite3 (instead of default 5s) to help work around https://github.com/adamgreig/sheepdog/issues/28

5.1.2 0.2.2

Released on 2014-10-27

- Fix a bug where multiple runs in the same Python session would break
- Count errors properly in verbose map mode
- Automatically call setrlimit when a mem_grab resource specifier is given

5.1.3 0.2.1

Released on 2014-07-10

- Documentation updates
- Remove redundant console output

5.1.4 0.2.0

Released on 2014-04-11

- Keep track of what servers are running to prevent duplicates being started
- Allow specification of pickle and marshal protocol versions
- Refactor __init__'s map_sync into map_async, get_results and map * *note that 'map_sync' is now 'map'*
- Handle task errors in the top level map and get_results functions

- Tidy examples somewhat

5.2 Version 0.1

Early work.

5.2.1 0.1.10

Released on 2014-04-04.

- Port is now selected at random by default (instead of 7676)
- Removed fear-specific default *ge_opts*
- Swaped to Marshall protocol 2 so 3.4 doesn't get sad, but still using Pickle protocol 3 so Py3 hosts won't be able to talk to Py2 workers yet
- Added *ssh_keyfile* option so a specific passphraseless SSH key can be used
- Added HTTP Basic Auth to HTTP requests

5.2.2 0.1.9

Released on 2014-04-04.

- Documentation improvements
- Actually release to PyPi, which got skipped for 0.1.8

5.2.3 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

5.2.4 0.1.7

Released on 2014-03-21.

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

5.2.5 0.1.6

Released on 2014-03-20.

- Fix tests for namespace serialisation.

5.2.6 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

5.2.7 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

5.2.8 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

5.2.9 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.

5.2.10 0.1.1

Released on 2013-12-04.

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

5.2.11 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 6

sheepdog

6.1 sheepdog Package

6.1.1 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__.**get_errors**(*request_id*, *dbfile*)
Fetch all the errors returned so-far for *request_id*.

sheepdog.__init__.**get_results**(*request_id*, *dbfile*, *block=True*, *verbose=False*)
Fetch results for *request_id*. If *block* is true, wait until all the results are in. Otherwise, return just what has been received so far.

If *verbose* is true, print a status message every second with the current number of results.

Returns a list of (arg, result) tuples.

Where an error occured or no result has been submitted yet, result will be None.

sheepdog.__init__.**map**(*f*, *args*, *config*, *ns=None*, *verbose=True*)
Submit *f* with each of *args* on GridEngine, wait until all the results are in, and return them in the same order as *args*. If an error occured for an arg, None is returned in that position. Call *get_errors* to get details on the errors that occured.

For details on *config*, see the documentation at: http://sheepdog.readthedocs.org/en/latest/configuration.html Or in docs/configuration.rst.

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

If *verbose* is true, print out how many results are in so-far while waiting.

sheepdog.__init__.**map_async**(*f*, *args*, *config*, *ns=None*)
Submit *f* with each of *args* on GridEngine, returning the (sheepdog-local) request ID.

For details on *config*, see the documentation at: http://sheepdog.readthedocs.org/en/latest/configuration.html Or in docs/configuration.rst.

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

6.1.2 `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*, *password*, *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.

`set_memlimit`(*fname='/home/docs/checkouts/readthedocs.org/user_builds/sheepdog/envs/latest/local/lib/python2.7/site-packages/Sheepdog-0.2.3-py2.7.egg/sheepdog/client.pyc'*)

`submit_results`()

6.1.3 `deployment` Module

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

class `sheepdog.deployment.`**`Deployer`**(*host*, *port*, *user*, *keyfile=None*)
Connect to a remote SSH server, copy a file over, run qsub.

__init__ takes (host, port, user, keyfile) 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.

6.1.4 `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*, *password*, *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.

password is the HTTP Basic Auth password to use when talking to *url*.

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.

6.1.5 `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*, *password*, *dbfile*)
Run the HTTP server for workers to request arguments and return results.

Should be used via get_server(port, password, dbfile) to manage servers running globally.

__init__ creates and starts the HTTP server.

`stop`()
Terminate the HTTP server.

`sheepdog.server.`**`check_auth`**(*username*, *password*)

`sheepdog.server.`**`get_config`**(**args*, ***kwargs*)
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_server`**(*port*, *password*, *dbfile*)
Either start a new server or retrieve a reference to an existing server. Only one server may run per port. If the server currently running on that port has a different password or dbfile, a RuntimeError is raised.

If *None* is specified for port, a port is picked randomly and that server is the one referenced for *None* thereafter.

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

`sheepdog.server.`**`report_error`**(**args*, ***kwargs*)
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.`**`requires_auth`**(*f*)

`sheepdog.server.`**`run_server`**(*port*, *password*, *dbfile*)
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`**(**args*, ***kwargs*)
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.

6.1.6 `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*)
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.

Use of ":memory:" is not advised as the web server runs in a separate process so will not share memory with the main interpreter process, making it rather difficult to retrieve results.

`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.

`get_tasks_with_results`(*request_id*)
Fetch all tasks for a given request_id, including results for all tasks where results have come in already.

Returns a list of (args, result) items in the order of the original args_list provided to new_request, where result may be None.

`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 7

Indices and tables

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

S

C

D

G

H

I

J

M

N

R

S