The Wholesome Goodness of Distributed Task Queues

Taking a little nibble on Django-Celery

Werner Fortmann

e: werner.fortmann@gmail.com

October 22, 2012

What is Django-Celery?

Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well.
The execution units, called tasks, are executed concurrently on a single or more worker servers using multiprocessing, Eventlet, or gevent. Tasks can execute asynchronously (in the background) or synchronously (wait until ready).
The example followed in this presentation is adapted from the one in the official South tutorial:

- `http://south.readthedocs.org/en/latest/tutorial/`

Initial Setup

1. Install your broker (we shall use Redis).

   ```
   sudo apt-get install redis-server
   ```

2. Install the Python API for redis into your python virtual environment.

   ```
   pip install redis
   ```

3. Install django-celery

   ```
   pip install django-celery
   ```

Now you are pretty much good to go. Next step, configure Django.

Configure Django

The steps to follow are:

- Add celery (“djcelery”) to your installed apps.
- Insert the djcelery.setup_loader code in settings.py.
- Add the celery config settings to settings.py.

```
######## celery (with Redis) settings  ################
BROKER_URL = "redis://localhost:6379/0"
CELERY_RESULT_BACKEND = "redis"
CELERY_REDIS_HOST = "localhost"
CELERY_REDIS_PORT = 6379
CELERY_REDIS_DB = 0
CELERY_IMPORTS = ("yourapp.tasks","yourotherapp.tasks", )
CELERYD_LOG_FILE='/tmp/celery-yourproject.log'
CELERY_TIMEZONE = 'Australia/Sydney'

import djcelery
djcelery.setup_loader()
```

- Sync, or prefereably migrate (using South), your database.

```
python manage.py migrate djcelery
```

Let's write some tasks.

With everything setup lets take a look at how you can write some tasks. For each app, I usually create a tasks module, to house all the tasks related to that particular app.
Our first task:

```
from celery.task import task

@task()
def add(x, y):
    result = x + y
    print('Executing task id %r, %r + %r = %r' % (
        add.request.id, x, y, result))
    return result
```

We now start our celery server:

```
/manage.py celeryd -v 2 -B -s celery -E -l INFO
```

And test by visiting `http://127.0.0.1:8000/djcelery/` Note: this isn't the simplest script, as it also starts the celery beat process. We'll get to this later.

Now for a useful task.

Ok let's look at a more useful task. Here is a task that queries an externally managed database to update time series data for a specified stock.
Our first task:

```
@task()
def ats_update_task(asset_exchg = None):
    """
    Update the ATS data for the specified asset_exchg
    """
    if asset_exchg is None:
        return

    from utils import fetch_and_save_price_frame

    fetch_and_save_price_frame(asset_exchg = asset_exchg,
      startdate = '1900-01-01')
```

Test again by visiting `http://127.0.0.1:8000/djcelery/`

Scheduling of periodic tasks.

Celery can also be used to schedule periodic tasks, using celery beat as the scheduler. This is what the -B option was for when we started the celery server.
Our first task:

```
from celery.decorators import periodic_task
from celery.schedules import crontab

@periodic_task(run_every =
crontab(hour="1", minute="00", day_of_week="Sunday"))
def ats_update_all():
    """
    Update all time series data
    """
    ATSUpdateAll()
```

Test by restarting the celery task server.

Where to next?

Well I'm pretty new to this... But I imagine firing signals would be a great way to work cleverly in an asynchronous way.

Who's got some suggestions?