A Python Task Queue Story

Why and how we migrated from Celery to RQ

Paris.py #2
22-07-2013

Sylvain Zimmer
@sylvinus

Task Queue?

- Offload long tasks to background workers
- Pattern used by almost all big companies
- PaaS support: Heroku, App Engine, ...
- Major implementations:
  - Gearman (Perl)
  - Resque (Ruby)
  - Celery (Python)

[x.taskqueue for x in startups]

- **Jamendo** (encoding MP3s, stats, emails, ...)
  - 2005-2010: Custom Python+MySQL task queue
  - 2010-present: Celery
- **Joshfire** (building apps with Eclipse, Xcode, ...)
  - 2011-present: Celery
- **Pricing Assistant** (crawling, machine learning, ...)
  - 2012-2013: Celery
  - Last month: Migrated to RQ!

Celery: Distributed Task Queue

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).

Celery is used in production systems to process millions of tasks a day.

Latest_news: **Celery 3.0 Released!**
on 7 Jun 2012, 6:17 p.m.

GETTING STARTED

Install celery by download or `pip install -U Celery`

Set up RabbitMQ, Redis or one of the other supported brokers

Select one of the following guides:

First steps with Python

First steps with Django

EASY TO INTEGRATE

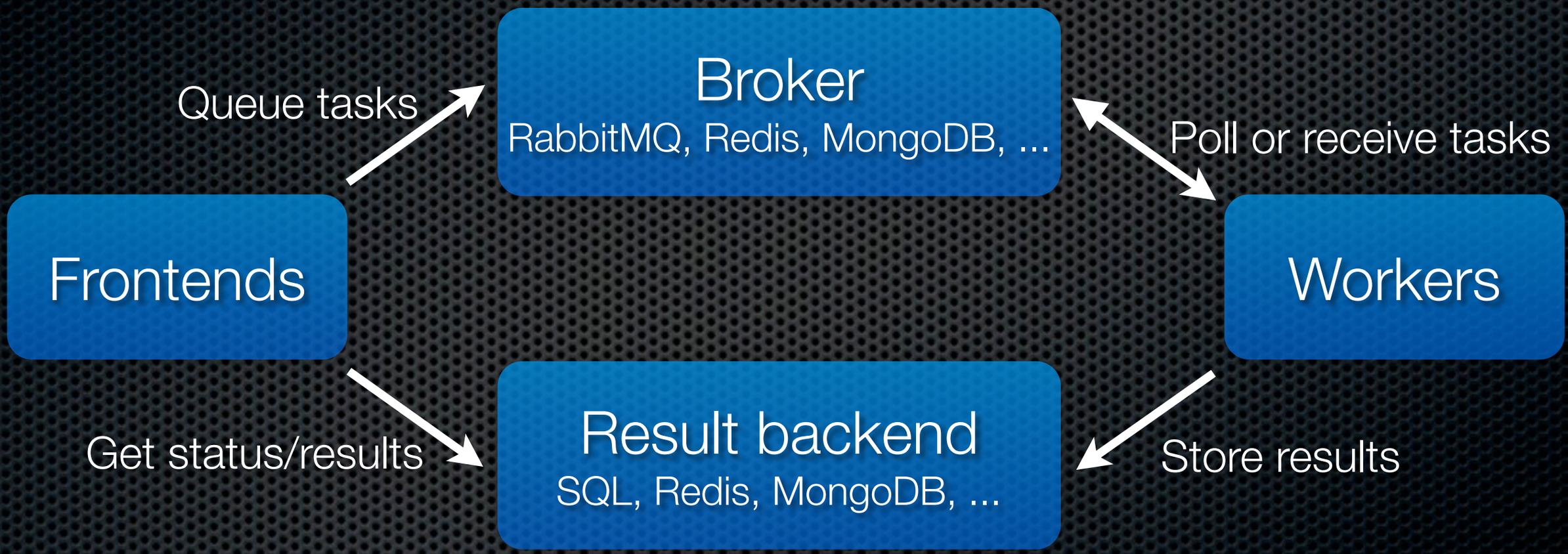
Celery is easy to integrate with web frameworks, some of which even have integration packages.

Celery is written in Python, but the protocol can be implemented in any language. It can also operate with other languages using webhooks.

MULTI BROKER SUPPORT

The recommended message broker is RabbitMQ, but support for Redis, Beanstalk, MongoDB, CouchDB, and databases (using SQLAlchemy or the Django ORM) is also available.

Celery design



Celery: the good

Easy start

```python
from celery import Celery

celery = Celery('tasks', broker='amqp://guest@localhost//')

@celery.task
def add(x, y):
    return x + y
```

```
>>> from tasks import add
>>> add.delay(4, 4)
```

It’s FAST

- Low latency thanks to RabbitMQ
- High throughput thanks to pre-fetching tasks
- *“Millions of tasks per minute per process”*

Countless features

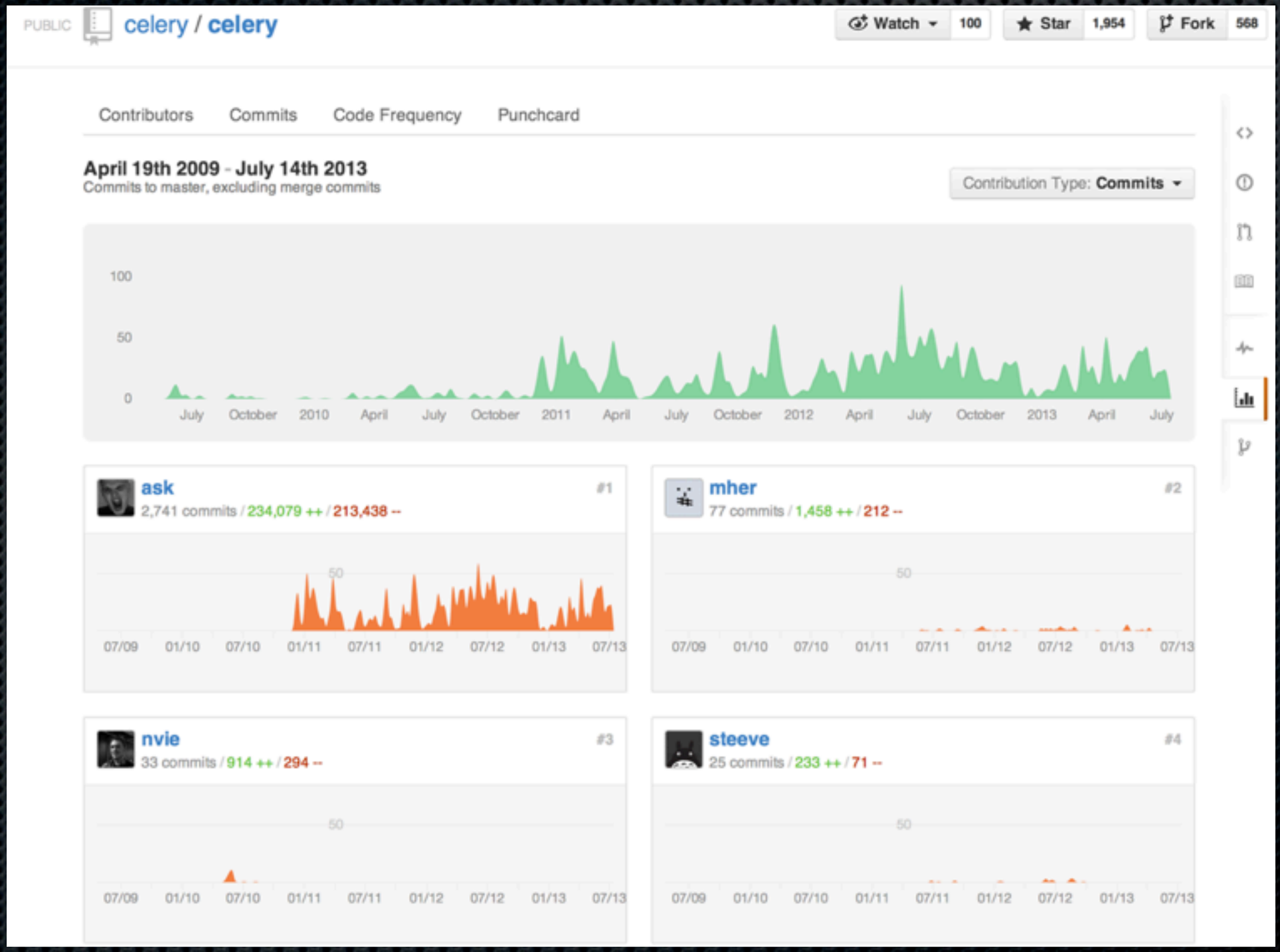
- Monitoring
- Workflows (grouping, chaining, ...)
- Time & rate limits
- Scheduling
- Autoscaling
- Autoreloading
- Customizable down to the core components

Other perks

- Well tested
- Pretty good documentation
- Main developer (@ask) is quite helpful
- Being the reference Python implementation

Celery: the bad

Work of one man



Complicated

```
def subclass_with_self(self, Class, name=None, attribute='app',
                       reverse=None, **kw):
    """Subclass an app-compatible class by setting its app attribute
    to be this app instance.

    App-compatible means that the class has a class attribute that
    provides the default app it should use, e.g.
    ``class Foo: app = None``.

    :param Class: The app-compatible class to subclass.
    :keyword name: Custom name for the target class.
    :keyword attribute: Name of the attribute holding the app,
                        default is 'app'.

    """
    Class = symbol_by_name(Class)
    reverse = reverse if reverse else Class.__name__

    def __reduce__(self):
        return _unpickle_appattr, (reverse, self.__reduce_args__())

    attrs = dict({attribute: self}, __module__=Class.__module__,
                 __doc__=Class.__doc__, __reduce__=__reduce__, **kw)

    return type(name or Class.__name__, (Class, ), attrs)
```

REALLY Complicated

```
def _install_stack_protection():
    # Patches BaseTask.__call__ in the worker to handle the edge case
    # where people override it and also call super.
    #
    # - The worker optimizes away BaseTask.__call__ and instead
    #   calls task.run directly.
    # - so with the addition of current_task and the request stack
    #   BaseTask.__call__ now pushes to those stacks so that
    #   they work when tasks are called directly.
    #
    # The worker only optimizes away __call__ in the case
    # where it has not been overridden, so the request/task stack
    # will blow if a custom task class defines __call__ and also
    # calls super().
    if not getattr(BaseTask, '_stackprotected', False):
        _patched['BaseTask.__call__'] = orig = BaseTask.__call__

        def __protected_call__(self, *args, **kwargs):
            stack = self.request_stack
            req = stack.top
            if req and not req._protected and \
                    len(stack) == 1 and not req.called_directly:
                req._protected = 1
                return self.run(*args, **kwargs)
            return orig(self, *args, **kwargs)
        BaseTask.__call__ = __protected_call__
        BaseTask._stackprotected = True
```

Complex

```
(venv)➜ site-packages git:(master) ✗ /usr/local/bin/sloccount billiard kombu celery
```

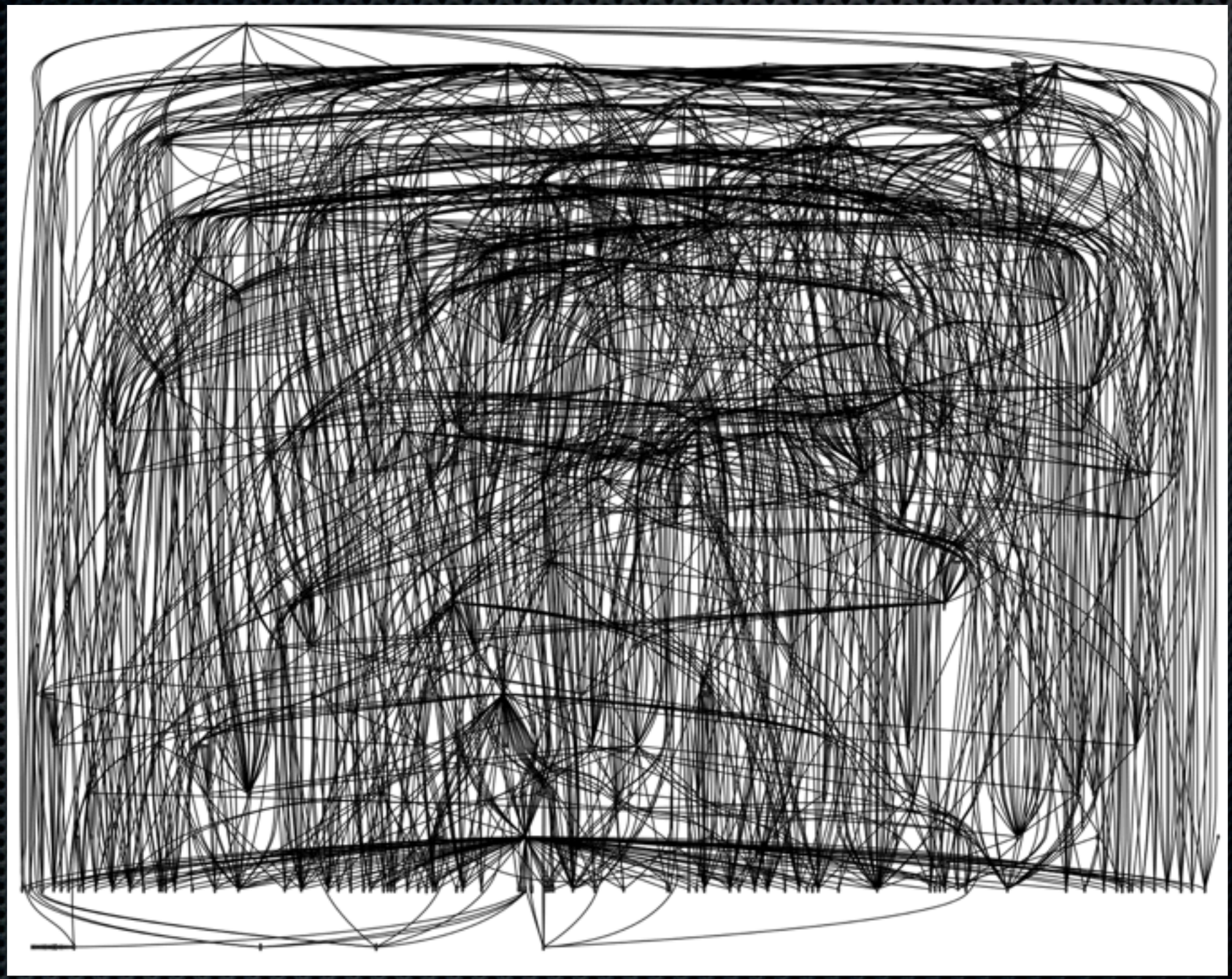
(after deleting tests)

```
Total Physical Source Lines of Code (SLOC)                = 27,598
Development Effort Estimate, Person-Years (Person-Months) = 6.52 (78.19)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 1.09 (13.10)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 5.97
Total Estimated Cost to Develop                           = $ 880,163
 (average salary = $56,286/year, overhead = 2.40).
```

(celery itself = 15,473)

/!\ Lines of code aren't a perfect metric

REALLY Complex



$ sfood celery | sfood-graph | dot -Tpdf

The ugly?

- There is no ugly.
- Be grateful for the open source software you use!
- Understand software before making a choice and blame yourself if you made the wrong one, not the authors!

Our paradoxical choice

- We needed *more* from our task queue
- We needed to understand 100% of how it works
- We didn’t need layers of abstraction for everything. Just one implementation that worked!

- = +

RQ

RQ (*Redis Queue*) is a simple Python library for queueing jobs and processing them in the background with workers. It is backed by Redis and it is designed to have a low barrier to entry. It should be integrated in your web stack easily.

RQ requires Redis >= 2.2.0.

Getting started

First, run a Redis server, of course. To put jobs on queues, you don't have to do anything special, just define your typically lengthy or blocking function:

```
import requests

def count_words_at_url(url):
    resp = requests.get(url)
    return len(resp.text.split())
```

Quick comparison

	Celery	RQ
Broker	RabbitMQ, Redis, MongoDB, ...	Redis
Concurrency	Master-slave processes	supervisord + fork()
Lines of code	~27k	~2k
Scheduler	Built-in	3rd-party
Routing	Advanced exchanges	List of queues

SIMPLE

- 3 main classes: Job, Queue, Worker
- Queue.enqueue_call() :

```
def enqueue_call(self, func, args=None, kwargs=None, timeout=None, result_ttl=None):
    """Creates a job to represent the delayed function call and enqueues
    it.

    It is much like `.enqueue()`, except that it takes the function's args
    and kwargs as explicit arguments.  Any kwargs passed to this function
    contain options for RQ itself.
    """
    timeout = timeout or self._default_timeout
    job = Job.create(func, args, kwargs, connection=self.connection,
                     result_ttl=result_ttl, status=Status.QUEUED)
    return self.enqueue_job(job, timeout=timeout)
```

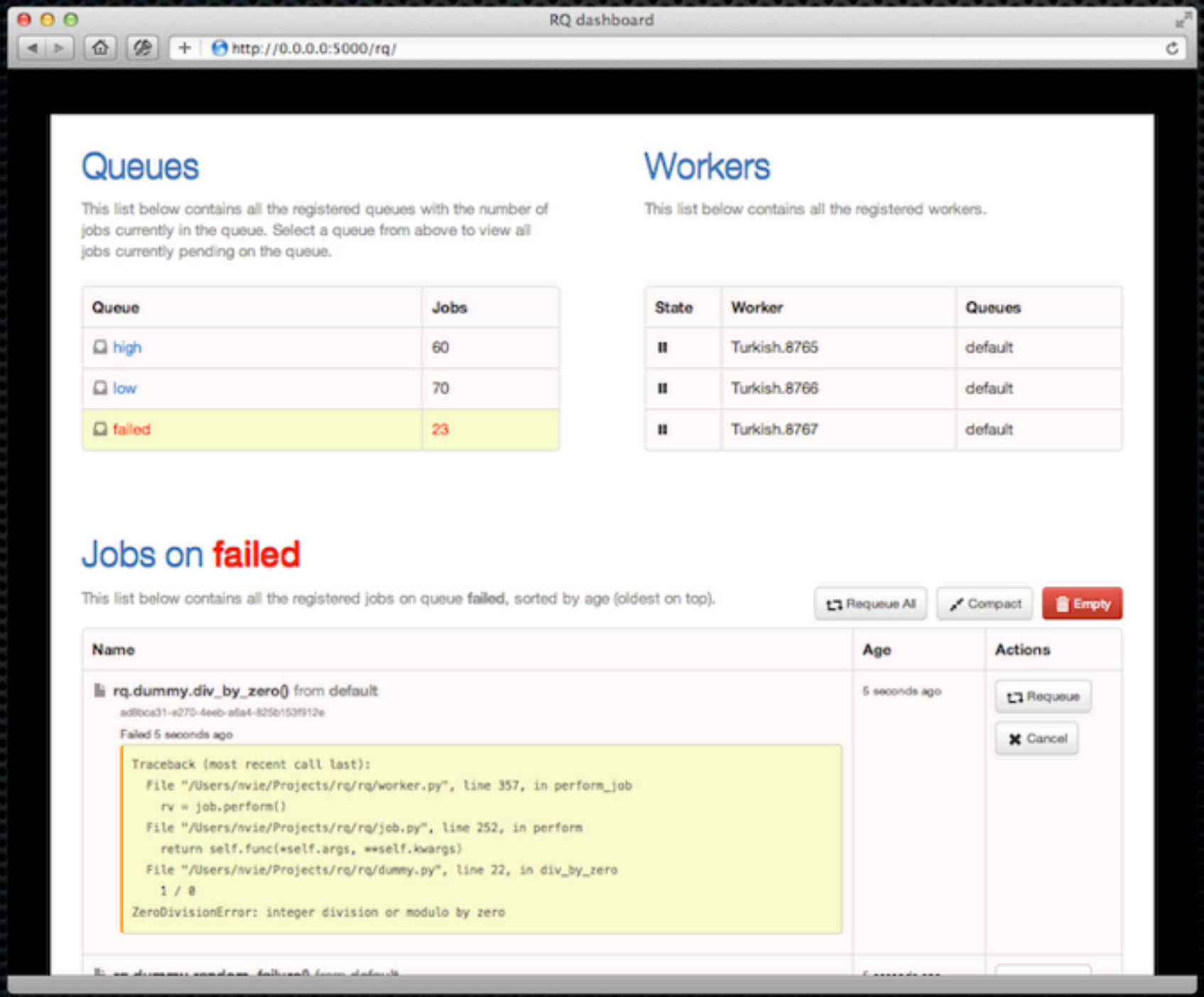
Still not perfect!

- Creator @nvie also the only primary author
- Much less mature (0.3.8) than Celery (3.0.21)
- Can’t serialize task results to JSON (only pickle)
- Limited concurrency model (for now)
- Monitoring tools can be improved

Our migration

- Surprisingly easy
- We didn't use much of the advanced Celery features
- Just changed task parent class + send_task()
- Joy of understanding 100% of your app's code ;)
- Discovered some long-standing errors in our tasks
- Only real step back was the monitoring frontend

Monitoring


RQ dashboard
http://0.0.0.0:5000/rq/
Queues
This list below contains all the registered queues with the number of jobs currently in the queue. Select a queue from above to view all jobs currently pending on the queue.
Queue
Jobs
high
60
low
70
failed
23
Workers
This list below contains all the registered workers.
State
Worker
Queues
Turkish.8765
default
Turkish.8766
default
Turkish.8767
default
Jobs on failed
This list below contains all the registered jobs on queue failed, sorted by age (oldest on top).
Requeue All
Compact
Empty
Name
Age
Actions
rq.dummy.div_by_zero() from default
ad8bca31-e270-4eeb-a6a4-825b153f912e
Failed 5 seconds ago
Traceback (most recent call last):
File "/Users/nvie/Projects/rq/rq/worker.py", line 357, in perform_job
rv = job.perform()
File "/Users/nvie/Projects/rq/rq/job.py", line 252, in perform
return self.func(*self.args, **self.kwargs)
File "/Users/nvie/Projects/rq/rq/dummy.py", line 22, in div_by_zero
1 / 0
ZeroDivisionError: integer division or modulo by zero
5 seconds ago
Requeue
Cancel

Monitoring improvements

Queues

Empty all

Queue	Jobs
default	1
finished	18230
parse	1145
started	78
failed	1034

Workers

State	Worker	Queues
▶ 16 ⏸ 0		highpriority, crawl, default, parse, find, filter, lowpriority
▶ 16 ⏸ 0		highpriority, default, lowpriority, find, parse, filter, crawl

Jobs Names

tasks.stats.competitorproduct.AllCompetitorStore	5808
tasks.stats.product.AllWithCompetitorProduct	924
tasks.stats.product.AllWithCompetitorStore	60
tasks.stats.store.All	1
tasks.stats.store.One	36

Jobs on failed

View all logs | Requeue all failed jobs | Compact | Empty | Cancel all

Name	Worker	Origin Queue	Args, logs, results	Status	Actions
tasks.crawler.parseurl.Product	c070d972-f954-4029-a525-536d5acea601.13	highpriority	Args : "{'competitorproduct': '514563cdd0ab560200001798', 'allow_automod': False})" Logs 12:58:03 INFO fetch.py:148 Fetching http://www.lecomptoirsante.com/	Created 1 hour ago Finished 1 hour ago FAILED	Requeue Cancel

https://github.com/pricingassistant/rq-dashboard

We're hiring Python hackers!

sylvain@pricingassistant.com

Thanks!

Questions?

sylvain@pricingassistant.com