Scrapy-Redis Documentation

Release 0.7.0-dev

Rolando Espinoza

Nov 13, 2017

Contents

Contents:

CHAPTER 1

Scrapy-Redis

Redis-based components for Scrapy.

- Free software: MIT license
- Documentation: https://scrapy-redis.readthedocs.org.
- Python versions: 2.7, 3.4+

1.1 Features

- Distributed crawling/scraping

  You can start multiple spider instances that share a single redis queue. Best suitable for broad multi-domain crawls.

- Distributed post-processing

  Scraped items gets pushed into a redis queued meaning that you can start as many as needed post-processing processes sharing the items queue.

- Scrapy plug-and-play components

  Scheduler + Duplication Filter, Item Pipeline, Base Spiders.

Note: This features cover the basic case of distributing the workload across multiple workers. If you need more features like URL expiration, advanced URL prioritization, etc., we suggest you to take a look at the Frontera project.

1.2 Requirements

- Python 2.7, 3.4 or 3.5
- Redis >= 2.8

- `Scrapy` >= 1.1
- `redis-py` >= 2.10

1.3 Usage

Use the following settings in your project:

```
# Enables scheduling storing requests queue in redis.
SCHEDULER = "scrapy_redis.scheduler.Scheduler"

# Ensure all spiders share same duplicates filter through redis.
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"

# Default requests serializer is pickle, but it can be changed to any module
# with loads and dumps functions. Note that pickle is not compatible between
# python versions.
# Caveat: In python 3.x, the serializer must return strings keys and support
# bytes as values. Because of this reason the json or msgpack module will not
# work by default. In python 2.x there is no such issue and you can use
# 'json' or 'msgpack' as serializers.
#SCHEDULER_SERIALIZER = "scrapy_redis.picklecompat"

# Don't cleanup redis queues, allows to pause/resume crawls.
#SCHEDULER_PERSIST = True

# Schedule requests using a priority queue. (default)
#SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.PriorityQueue'

# Alternative queues.
#SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.FifoQueue'
#SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.LifoQueue'

# Max idle time to prevent the spider from being closed when distributed crawling.
# This only works if queue class is SpiderQueue or SpiderStack,
# and may also block the same time when your spider start at the first time (because␣
↪the queue is empty).
#SCHEDULER_IDLE_BEFORE_CLOSE = 10

# Store scraped item in redis for post-processing.
ITEM_PIPELINES = {
    'scrapy_redis.pipelines.RedisPipeline': 300
}

# The item pipeline serializes and stores the items in this redis key.
#REDIS_ITEMS_KEY = '%(spider)s:items'

# The items serializer is by default ScrapyJSONEncoder. You can use any
# importable path to a callable object.
#REDIS_ITEMS_SERIALIZER = 'json.dumps'

# Specify the host and port to use when connecting to Redis (optional).
#REDIS_HOST = 'localhost'
#REDIS_PORT = 6379

# Specify the full Redis URL for connecting (optional).
# If set, this takes precedence over the REDIS_HOST and REDIS_PORT settings.
```

```
#REDIS_URL = 'redis://user:pass@hostname:9001'

# Custom redis client parameters (i.e.: socket timeout, etc.)
#REDIS_PARAMS  = {}
# Use custom redis client class.
#REDIS_PARAMS['redis_cls'] = 'myproject.RedisClient'

# If True, it uses redis' ``SPOP`` operation. You have to use the ``SADD``
# command to add URLs to the redis queue. This could be useful if you
# want to avoid duplicates in your start urls list and the order of
# processing does not matter.
#REDIS_START_URLS_AS_SET = False

# Default start urls key for RedisSpider and RedisCrawlSpider.
#REDIS_START_URLS_KEY = '%(name)s:start_urls'

# Use other encoding than utf-8 for redis.
#REDIS_ENCODING = 'latin1'
```

Note: Version 0.3 changed the requests serialization from `marshal` to `cPickle`, therefore persisted requests using version 0.2 will not able to work on 0.3.

1.4 Running the example project

This example illustrates how to share a spider's requests queue across multiple spider instances, highly suitable for broad crawls.

1. Setup scrapy_redis package in your PYTHONPATH
2. Run the crawler for first time then stop it:

```
$ cd example-project
$ scrapy crawl dmoz
... [dmoz] ...
^C
```

3. Run the crawler again to resume stopped crawling:

```
$ scrapy crawl dmoz
... [dmoz] DEBUG: Resuming crawl (9019 requests scheduled)
```

4. Start one or more additional scrapy crawlers:

```
$ scrapy crawl dmoz
... [dmoz] DEBUG: Resuming crawl (8712 requests scheduled)
```

5. Start one or more post-processing workers:

```
$ python process_items.py dmoz:items -v
...
Processing: Kilani Giftware (http://www.dmoz.org/Computers/Shopping/Gifts/)
Processing: NinjaGizmos.com (http://www.dmoz.org/Computers/Shopping/Gifts/)
...
```

1.5 Feeding a Spider from Redis

The class *scrapy_redis.spiders.RedisSpider* enables a spider to read the urls from redis. The urls in the redis queue will be processed one after another, if the first request yields more requests, the spider will process those requests before fetching another url from redis.

For example, create a file *myspider.py* with the code below:

```
from scrapy_redis.spiders import RedisSpider

class MySpider(RedisSpider):
    name = 'myspider'

    def parse(self, response):
        # do stuff
        pass
```

Then:

1. run the spider:

   ```
   scrapy runspider myspider.py
   ```

2. push urls to redis:

   ```
   redis-cli lpush myspider:start_urls http://google.com
   ```

Note: These spiders rely on the spider idle signal to fetch start urls, hence it may have a few seconds of delay between the time you push a new url and the spider starts crawling it.

CHAPTER 2

Installation

2.1 Stable release

To install Scrapy-Redis, run this command in your terminal:

```
$ pip install scrapy-redis
```

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Scrapy-Redis can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/rolando/scrapy-redis
```

Or download the tarball:

```
$ curl  -OL https://github.com/rolando/scrapy-redis/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ pip install -e .
```

CHAPTER 3

History

3.1 0.7.0-dev (unreleased)

- Unreleased.

3.2 0.6.8 (2017-02-14)

- Fixed automated release due to not matching registered email.

3.3 0.6.7 (2016-12-27)

- Fixes bad formatting in logging message.

3.4 0.6.6 (2016-12-20)

- Fixes wrong message on dupefilter duplicates.

3.5 0.6.5 (2016-12-19)

- Fixed typo in default settings.

3.6 0.6.4 (2016-12-18)

- Fixed data decoding in Python 3.x.

- Added `REDIS_ENCODING` setting (default `utf-8`).
- Default to `CONCURRENT_REQUESTS` value for `REDIS_START_URLS_BATCH_SIZE`.
- Renamed queue classes to a proper naming conventiong (backwards compatible).

3.7 0.6.3 (2016-07-03)

- Added `REDIS_START_URLS_KEY` setting.
- Fixed spider method `from_crawler` signature.

3.8 0.6.2 (2016-06-26)

- Support `redis_cls` parameter in `REDIS_PARAMS` setting.
- Python 3.x compatibility fixed.
- Added `SCHEDULER_SERIALIZER` setting.

3.9 0.6.1 (2016-06-25)

- **Backwards incompatible change:** Require explicit `DUPEFILTER_CLASS` setting.
- Added `SCHEDULER_FLUSH_ON_START` setting.
- Added `REDIS_START_URLS_AS_SET` setting.
- Added `REDIS_ITEMS_KEY` setting.
- Added `REDIS_ITEMS_SERIALIZER` setting.
- Added `REDIS_PARAMS` setting.
- Added `REDIS_START_URLS_BATCH_SIZE` spider attribute to read start urls in batches.
- Added `RedisCrawlSpider`.

3.10 0.6.0 (2015-07-05)

- Updated code to be compatible with Scrapy 1.0.
- Added *-a domain=...* option for example spiders.

3.11 0.5.0 (2013-09-02)

- Added *REDIS_URL* setting to support Redis connection string.
- Added *SCHEDULER_IDLE_BEFORE_CLOSE* setting to prevent the spider closing too quickly when the queue is empty. Default value is zero keeping the previous behavior.
- Schedule preemptively requests on item scraped.
- This version is the latest release compatible with Scrapy 0.24.x.

3.12 0.4.0 (2013-04-19)

- Added *RedisSpider* and *RedisMixin* classes as building blocks for spiders to be fed through a redis queue.
- Added redis queue stats.
- Let the encoder handle the item as it comes instead converting it to a dict.

3.13 0.3.0 (2013-02-18)

- Added support for different queue classes.
- Changed requests serialization from *marshal* to *cPickle*.

3.14 0.2.0 (2013-02-17)

- Improved backward compatibility.
- Added example project.

3.15 0.1.0 (2011-09-01)

- First release on PyPI.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search