Scrapy: General Message Queues as Storage for Requests

Code contributions

Contributions to Scrapy

<table>
<thead>
<tr>
<th>Issue</th>
<th>PR</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>#50</td>
<td>#4413</td>
<td>Ignore a domain in allowed_domains with port and issue a warning</td>
<td>Merged</td>
</tr>
<tr>
<td></td>
<td>#4410</td>
<td>Fix handling of None in allowed_domains</td>
<td>Merged</td>
</tr>
<tr>
<td>#4007</td>
<td></td>
<td>Fix DummyStatsCollector</td>
<td>Merged</td>
</tr>
<tr>
<td></td>
<td>#2531</td>
<td>Small documentation fix</td>
<td>Merged</td>
</tr>
</tbody>
</table>

Contributions involving Scrapy

PyFeeds

PyFeeds is an open source project based on the Scrapy framework that allows a user to generate RSS feeds for arbitrary websites. I created PyFeeds almost four years ago together with a friend because less and less websites support RSS feeds.

- Source code: https://github.com/pyfeeds/pyfeeds
- Documentation: https://pyfeeds.readthedocs.io/en/latest/

Blog post about handling certain types of XML documents in Scrapy


Project information

- **Sub-org name**: Scrapinghub
- Project idea: http://gsoc2020.scrapinghub.com/ideas#message-queues
- Corresponding GitHub issue: https://github.com/scrapy/scrapy/issues/4326
Project abstract
Scrapy uses queues for handling requests. The scheduler pushes requests to the queue and pops them from the queue when the next request is ready to be processed. At the moment, there is no support for external message queues (e.g. Redis¹, Kafka, etc.) implemented in Scrapy, however, there are external libraries (https://github.com/rmax/scrapy-redis and others) that bridge Scrapy with external message queues. The goal of this project is to allow users to use external message queues out of the box without having to install a bridge like scrapy-redis. [1]

Detailed description

Status quo
The scheduler uses queues to enqueue requests and to decide what request to schedule next. The scheduler does not use the queue implementations directly but instead, the queue implementation is wrapped by a priority queue (this pattern is called Decorator Pattern by the “Gang of Four” book [3]). This way it is possible to combine different priority queue classes with different queue implementation classes.

¹While Redis is not a message queue, it is a simple key value datastore that can be used like a primitive message queue. Therefore Redis will be described as a message queue in this document.
The scheduling architecture that makes use of queues looks as follows:

A priority queue is created for a memory-backed queue and, if the JOBDIR setting is set, a disk-backed queue as well. The optional disk-based queue has the advantage that scraping can be stopped and continued at any time but requests must be serialized and deserialized when being pushed to or popped from the queue (which might not always work). In such cases, the memory queue works as a fallback.

A new type of disk-based queues: External queues

The goal, as stated in the project abstract, is to support a special type of disk-based queues: external queues. In the first iteration, the queue will be used for storing and retrieving requests. Meta data will still be saved on disk in a directory set by the JOBDIR folder. If the setting SCHEDULER_DISK_QUEUE is set to class, e.g. scrapy.squeues.PickleFifoRedisQueue, then that class is initialized, possibly with some settings so that the connection can be established. The class itself can use the LPUSH and RPOP Redis commands to implement a queue under the hood (similar to what scrapy-redis already does). [4] Support for different types of external queues like Kafka can be implemented in a similar way.

External queues without the JOBDIR folder

In the next step, the possibility of getting rid of the JOBDIR folder if an external queue is used should be evaluated. The goal is to not store information on disk anymore and only in the external data store Redis.

Request duplicate filtering with external data stores

Request duplicate filtering is currently based on adding fingerprints to an in-memory set. In the case of a disk-based queue, fingerprints are saved on disk as well. Requests could be either only saved in an external data store (Redis) or saved in memory and in Redis/on disk.

Distributed crawls

Using a shared external data store would allow to distribute crawls easily on multiple machines, i.e. it would allow to run Scrapy on multiple machines with a distributed workload. This is also one of the features of scrapy-redis where it is implemented with a custom request duplicate filter.
Support for other external queues
As time permits, support for other queues, e.g. Kafka and RabbitMQ, could be implemented as well.

Weekly timeline

- **Community Bonding** (April 27 - May 17): Reading up on Redis and other message queues
- **Week 1** (May 18-22): Implementing a new disk-based queue with Redis as a message queue backend
- **Week 2** (May 25-29): Evaluating the possibility of disk-less external queues
- **Week 3** (June 1): Refactoring Redis implementation; incorporating feedback
- **Week 4** (June 8): Writing tests, documentation
- **Week 5** (June 15): Implementing duplicate filtering with external data sources
- **Week 6** (June 22): Tests, fixes and documentation
- **Week 7** (June 29): Implementing distributed crawling using common message queues
- **Week 8** (July 6): Tests, fixes and documentation
- **Week 9** (July 13): Implementing of other message queue (Kafka)
- **Week 10** (July 20): Implementing of other message queue (RabbitMQ)
- **Week 11** (July 27): Implementing of other message queue (to be decided)
- **Week 12** (August 3): Final fixing and cleanup of code, tests and documentation
- **Final week** (August 10): Pull request is reviewed, feedback incorporated and ready to be merged

Other commitments
Exams are scheduled for beginning to mid of May and will not impair my ability to work on the project. I do not have any commitments from June to August.

References