kombu_redis_priority Documentation

Release 0.2.0

Shaon Barman, AJ Renold, Yoriyasu Yano

May 08, 2018

Contents:

kombu_redis_priority is a plugin into kombu that provides a Redis backed transport backend that supports prioritized ordering of messages through SortedSet.

kombu_redis_priority works by extending the existing redis transport backend and replacing references to the list data type with sortedset. It then utilizes the priority interface provided in AMQP to order the messages within the sortedset, thereby allowing implementations of prioritized messaging.

In addition, it takes advantage of lexicographical ordering of messages to achieve FIFO queueing when the priorities are equivalent. This is achieved by prefixing message jsons with epoch timestamps at the time of enqueueing messages into the sortedset.

CHAPTER 1

Installation

To get started using the redis_priority transport:

First, install the package:

```
pip install kombu-redis-priority
```

Then, import the package at the start of your application, before you start configuring kombu. For example in a Celery application, you would add the following line before you configure your celery application in a celery.py file:

```
import kombu_redis_priority.transport.redis_priority_async
app = Celery('redis_priority_example')
```

You can now use the redis_priority transport by referring to the *redisprioritysync* transport wherever you configure kombu.

Example:

```
BROKER_URL = 'redisprioritysync://{}:{}/{}'.format(REDIS_HOST, REDIS_PORT, REDIS_
↪QUEUE)
```

CHAPTER 2

Usage

After configuring kombu-redis-priority, you can use it with Celery when applying tasks:

```
your_task.apply_async(zpriority=1000)
```

The parameter zpriority is used instead of Celery's priority parameter to avoid confusion with other Celery priority implementations. Lower values will have higher priority.

The default zpriority assigned to tasks is the lowest priority - +inf

CHAPTER 3

Queue Scheduling

By default the *redis_priority* transport will consume from multiple queues in a round robin fashion, like all the other transports for *kombu*. Currently, this transport does not support the other strategies specified in *kombu* (*priority* and *sorted*). Instead, this transport provides the *prioritized_levels* strategy described below.

3.1 Prioritized Scheduling

Given a configuration of queues encoded as:

```
{
    LEVEL: [QUEUE]
}
```

where *LEVEL* is a numeric value indicating priority (smaller first) and *[QUEUE]* indicates a list of queue names, the scheduler will walk from smallest level to highest level, only advancing levels if the smaller levels are empty. Within levels, the queues are rotated in round robin fashion.

To honor the prefernece for the lower levels, we will move back to the lowest level when we do a full rotation of the current level.

You can configure the *redis_priority* transport to use this method by using the *queue_order_strategy* and *prioritized_levels_queue_config* transport options, configured with *BROKER_TRANSPORT_OPTIONS*.

Example:

```
BROKER_TRANSPORT_OPTIONS = {
    'queue_order_strategy': 'prioritized_levels',
    'prioritized_levels_queue_config': {
        0: ['TimeMachine', 'FluxCapacitor'],
        1: ['1985', '1955', '2015']
    }
}
```

Note that any queue that the worker is specified to consume which is not in the *prioritized_levels_queue_config* is automatically specified at the highest level (max int).

CHAPTER 4

Testing

kombu_redis_priority uses Travis for CI builds and coveralls for coverage reporting.

You can also run the tests locally using setuptools:

```
python setup.py test
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search