tarantool-queue Documentation

Release 0.1.4

bigbes

October 29, 2015

Contents

Python Bindings for Tarantool Queue.

Library depends on:

- msgpack-python
- tarantool

Basic usage can be found in tests. Description on every command is in source code.

Big thanks to Dmitriy Shveenkov and Alexandr (FZambia) Emelin.

For install of latest "stable" version type:

```
# using pip
$ sudo pip install tarantool-queue
# or using easy_install
$ sudo easy_install tarantool-queue
# or using python
$ wget http://bit.ly/tarantool_queue -O tarantool_queue.tar.gz
$ tar xzf tarantool_queue.tar.gz
$ cd tarantool-queue-{version}
$ sudo python setup.py install
```

For install bleeding edge type:

```
$ sudo pip install git+https://github.com/tarantool/tarantool-queue-python.git
```

For configuring Queue in Tarantool read manual Here or read *Prepare server*.

Then just **import** it, create **Queue**, create **Tube**, **put** and **take** some elements:

```
>>> from tarantool_queue import Queue
>>> queue = Queue("localhost", 33013, 0)
>>> tube = queue.tube("name_of_tube")
>>> tube.put([1, 2, 3])
Not taken task instance
>>> task = tube.take()
>>> task.data # take task and read data from it
[1, 2, 3]
>>> task.ack() # move this task into state DONE
True
```

That's all, folks!

CHAPTER 1

See Also

- Documentation
- Repository

1.1 Queue API

Basic Definitions:

- ttl - Time to Live of task.
- ttr - Time to Release of task.
- pri - Priority of task.
- delay - Delay for task to be added to queue.

Warning: Don't use constructor of Task and Tube. Task's are created by Tube and Queue methods. For creating Tube object use *Queue.tube(name)*

class tarantool_queue.**Queue**(*host='localhost'*, *port=33013*, *space=0*, *schema=None*)

Tarantool queue wrapper. Surely pinned to space. May create tubes. By default it uses msgpack for serialization, but you may redefine serialize and deserialize methods. You must use Queue only for creating Tubes. For more usage, please, look into tests. Usage:

```
>>> from tarantool_queue import Queue
>>> queue = Queue()
>>> tube1 = queue.create_tube('holy_grail', ttl=100, delay=5)
# Put task into the queue
>>> tube1.put([1, 2, 3])
# Put task into the beggining of queue (Highest priority)
>>> tube1.urgent([2, 3, 4])
>>> tube1.get() # We get task and automaticaly release it
>>> task1 = tube1.take()
>>> task2 = tube1.take()
>>> print(task1.data)
    [2, 3, 4]
>>> print(task2.data)
    [1, 2, 3]
>>> del task2
>>> del task1
>>> print(tube1.take().data)
```

```
    [1, 2, 3]
# Take task and Ack it
>>> tube1.take().ack()
    True
```

DataBaseError
alias of DatabaseError

exception NetworkError(*orig_exception=None*, **args*)
Error related to network

Queue.**deserialize**
Deserialize function: must be Callable. If sets to None or delete, then it will use msgpack for deserializing.

Queue.**peek**(*task_id*)
Return a task by task id.

Parameters **task_id** (*string*) – UUID of task in HEX

Return type *Task* instance

Queue.**serialize**
Serialize function: must be Callable. If sets to None or deleted, then it will use msgpack for serializing.

Queue.**statistics**(*tube=None*)
Return queue module statistics accumulated since server start. Output format: if tube != None, then output is dictionary with stats of current tube. If tube is None, then output is dict of t stats, ...} e.g.:

```
    >>> tube.statistics()
    # or queue.statistics('tube0')
    # or queue.statistics(tube.opt['tube'])
    {'ack': '233',
    'meta': '35',
    'put': '153',
    'release': '198',
    'take': '431',
    'take_timeout': '320',
    'tasks': {'buried': '0',
            'delayed': '0',
            'done': '0',
            'ready': '0',
            'taken': '0',
            'total': '0'},
    'urgent': '80'}
    or
    >>> queue.statistics()
    {'tube0': {'ack': '233',
            'meta': '35',
            'put': '153',
            'release': '198',
            'take': '431',
            'take_timeout': '320',
            'tasks': {'buried': '0',
                    'delayed': '0',
                    'done': '0',
                    'ready': '0',
                    'taken': '0',
                    'total': '0'},
            'urgent': '80'}}
```

Parameters **tube** (*string or None*) – Name of tube

Return type dict with statistics

Queue.**tarantool_connection**
Tarantool Connection class: must be class with methods call and __init__. If it sets to None or deleted - it will use the default tarantool.Connection class for connection.

Queue.**tarantool_lock**
Locking class: must be locking instance with methods __enter__ and __exit__. If it sets to None or delete - it will use default threading.Lock() instance for locking in the connecting.

Queue.**truncate**(*tube*)
Truncate queue tube, return quantity of deleted tasks

Parameters **tube** (*string*) – Name of tube

Return type int

Queue.**tube**(*name*, ***kwargs*)
Create Tube object, if not created before, and set kwargs. If existed, return existed Tube.

Parameters

- **name** (*string*) – name of Tube
- **delay** (*int*) – default delay for Tube tasks (Not necessary, will be 0)
- **ttl** (*int*) – default TTL for Tube tasks (Not necessary, will be 0)
- **ttr** (*int*) – default TTR for Tube tasks (Not necessary, will be 0)
- **pri** (*int*) – default priority for Tube tasks (Not necessary)

Return type *Tube* instance

1.2 Quick Start

1.2.1 Prepare server

1. Install tarantool on your favourite OS. For more information, please refer to User Guide (Section: Downloading and installing a binary package).
2. Download tarantool.cfg and init.lua from Queue repo

```
$ wget https://raw.github.com/tarantool/queue/master/tarantool.cfg # Download configuration file for
$ wget https://github.com/tarantool/queue/blob/master/init.lua # Download queue script
$ tarantool_box --init-storage # Init tarantool storage files
$ tarantool_box # Start tarantool
```

Done!

1.2.2 Install tarantool-queue-python

```
# using pip
$ sudo pip install tarantool-queue
# or using easy_install
$ sudo easy_install tarantool-queue
# or using python
```

```
$ wget http://bit.ly/tarantool_queue -O tarantool_queue.tar.gz
$ tar xzf tarantool_queue.tar.gz
$ cd tarantool-queue-{version}
$ sudo python setup.py install
```

1.2.3 Connecting to server and basic operations

In the beggining you must **import tarantool-queue** and create **Queue** object:

```
from tarantool_queue import Queue
queue = Queue('localhost', 33020, 0)
```

Queue object is an aggregator for Tubes: Tube is the queue. When you put task into queue, you associate name of tube with it. When you take task, you take if from some tube. For the beggining you must create Tube object with **Queue.tube(name)** method. Basic operations for Tubes are: **Tube.put(task)** and **Tube.get()** When you have done all you want with this task you must make **Task.ack()** it or **Task.release()** it back to the queue.

```
# On producer:
appetizers = queue.tube('appt-s')
appetizers.put('Egg-Bacon') # put meal
appetizers.put('Egg-Sausage-Bacon')
appetizers.put('Egg and Spam')
appetizers.put('Egg-Bacon and Spam')
appetizers.put('Egg-Bacon-Sausage')
appetizers.put('Spam-Bacon-Sausage-Spam')
appetizers.put('Spam-Egg-Spam-Spam-Bacon-Spam')
appetizers.put('Spam-Spam-Spam-Egg-Spam')
# Spam, Spam, Spam, Spam... Lovely Spam! Wonderful Spam!
...
```

```
# On consumer number 1 (Viking):
appetizers = queue.tube('appt-s')
meal = appetizers.take(30) # wait for 'meal' in blocking mode for 30 seconds
while meal is not None:
    if meal.data.find('Spam') == -1: # we don't want meal without spam
        meal.release(delay=1)
    else:
        eat(meal.data) # do something
        meal.ack() # acknowlege, that you did all you want with this 'meal'
    meal = appetizers.take(30) # take next meal
exit(1) # no meal for 30 seconds, go out from here
```

```
# On consumer number 2 (Missus):
appetizers = queue.tube('appt-s')
meal = appetizers.take(30) # wait for 'meal' in blocking mode for 30 seconds
while meal is not None:
    if meal.data.find('Spam') != -1: # she is tired from spam
        meal.release(delay=1)
    else:
        eat(meal.data) # do something
        meal.ack() # acknowlege, that you did all you want with this 'meal'
    meal = appetizers.take(30) # take next meal
exit(1) # no meal for 30 seconds, go out from here
```

What if we forget to ack or release the task?

Task class has destructor, that automaticly releases the task, if you have done nothing with it. e.g:

```
# You're consumer of some great spam:
def eat_spam(tube):
    meal = tube.take()
    if (meal.data.find('Spam') != -1)
        meal.ack()
        consume(meal) # do_something
    return # oops! we forget to release task if it has not spam in it!
    # but that's ok, GC will do it when his time will come.
```

What data we can push into tubes?

Queue uses msgpack (It's like JSON. but fast and small) for default *serializing* of data, so by default you may *serialize* **dicts**, **tuples/lists**, **strings**, **numbers** and **others basic types**.

If you want to push another objects to Tubes you may define another *serializers*. By default *serializers* of Tubes are None and it uses Queue *serializer*. If you set Tube *serializer* to callable object it will use it, instead of Queue *serializer*. e.g.:

```
import bz2
import json
import pickle

from tarantool_queue import Queue

queue = Queue('localhost', 33020, 0)

jsons = queue.tube('json')
jsons.serialize =   (lambda x: json.dumps(x)) # it's not necessary to use lambda in your projects
jsons.deserialize = (lambda x: json.loads(x)) # but object, that'll serialize and deserialize must be

pickls = queue.tube('pickle')
pickls.serialize =   (lambda x: pickle.dump(x))
pickls.deserialize = (lambda x: pickle.load(x))

bz2s = queue.tube('bz2')
bz2s.serialize =   (lambda x: bz2.compress(json.dumps(x)))
bz2s.deserialize = (lambda x: json.loads(bz2.decompress(x)))

default = queue.tube('default')

jsons.put([1, 2, 3])  # it will put [1, 2, 3] in json into queue.
pickls.put([2, 3, 4]) # it will pickle [2, 3, 4] and put it into queue.
bz2.put([3, 4, 5])    # it will bzip' [3, 4, 5] in json and put it into queue.

default.put([4, 5, 6]) # msgpack will pack it and put into queue.
queue.serialize =   (lambda x: pickle.dump(x))
queue.deserialize = (lambda x: pickle.load(x))
default.put([4, 5, 6]) # but now it'll be pickled.

# to reset serializers you must simply assign None to serializer:
queue.serialize =   None # it will restore msgpack as serializer
queue.deserialize = None # it will restore msgpack as deserializer
bz2s.serialize =   None # it will tell python to use Queue serializer(msgpack) instead of bz2
```

```
bz2s.deserialize = None # it will tell python to use Queue deserializer(msgpack) instead of bz2
default.put([4, 5, 6]) # msgpack will pack it again.
```

But, i have very important task that needs to be done!

It's OK! You must use **Tube.urgent(data)**!

```
appetizers = queue.tube('appt-s')
appetizers.put('Egg-Bacon') # put meal
appetizers.put('Egg-Sausage-Bacon') # another boring meal
appetizers.urgent('Spam-Egg-Spam-Spam-Bacon-Spam') # very very tasty meal with a lot of SPAM

meal1 = appetizers.take() ; print meal1.data # Spam-Egg-Spam-Spam-Bacon-Spam
meal2 = appetizers.take() ; print meal2.data # Egg-Bacon
meal3 = appetizers.take() ; print meal3.data # Egg-Sausage-Bacon

meal1.ack() ; meal2.ack() ; meal3.ack()
```

Ok! But i've some spam today! I want to know how much.

```
appetizers = queue.tube('appt-s')
appetizers.statistics() # will show you how many spam you've 'baked' and 'sold'
queue.statistics() # will show you overall stats of your cafe
```

I have some spam, that is so awfully bad, that i want to bury deep inside.

```
appetizers = queue.tube('appt-s')
task = appetizers.get()
task.bury() # it will bury meal deep inside
task.dig() # it will 'unbury' meal, if you'll need it in future.
task.delete() # it will destroy your 'meal' once and for all.
appetizers.kick(number) # it will 'unbury' a number of tasks in this Tube.
task.done('New great SPAM with SPAM and HAM') # or you may replace this 'meal' with another.
```

But *Task.release()* returns task into the beggining! I want it to be in the end!

Simply use **Task.requeue()** instead of **Task.release()**!

SUDDENLY I have UUID of my 'meal', and i REALLY REALLY want this meal. What should i do?

You must use **Queue.peek(uuid)** method!

```
appetizers = queue.tube('appt-s')
meal_uuid = '550e8400-e29b-41d4-a716-446655440000'
task = queue.peek(meal_uuid)
print task.data # Spam-Egg-Spam-Spam-Bacon-Spam
```

Question-Answer

Q. What should i do, to use my own great tarantool connector in this Queue? How may i reset it into defaults?

A. You must simply use **Queue.tarantool_connector** field for setting it. Just ensure that your connector has **constructor** and **call** fields.

For reseting it simply do:

```
del(queue.tarantool_connector)
# OR
queue.tarantool_connector = None
```

Q. I'm using another great coroutines library! I really need another locking mechanism, instead of your threading.Lock.

A. It's ok! You may simply set **Queue.tarantool_lock** field with your lock. Just assure that your locking mechanism has **__enter__** and **__exit__** methods (your lock will be used in the "with LOCK:..." construction)

For reseting it simply do:

```
del(queue.tarantool_lock)
# OR
queue.tarantool_lock = None
```

And Now for Something Completely Different..

t

D

P

Q

S

T