Message Queueing

20 March 2015

Message Queueing--How it works

Publisher
One or more publishers, e.g.: a script gives the middleware a compound query which is vetted and published to the queue as lower-priority jobs. This is called a *topic* (as opposed to a *pure queue* which is a *point-to-point* concept not covered here).

Other points in the system submit jobs at normal priority.

Consumer
One or more consumers, i.e.: a search server, listen for work in the queue taking jobs that are compound queries and run them.

And vice-versa...
In turn, a search server becomes a publisher of a different topic which is the completed query and any result. Interested consumers pull these results from the queue.

How it works (continued)

Where do you put your “stuff” when you’ve finished?

Typically, messages have a length-limit, but can usually accommodate considerable data.
They can simply hold “pointers” or references to any other place including database oids, filesystem paths, GUIDs, etc.

However, some systems have a message longevity problem. Sometimes this is configurable, sometimes not.

Other problems or surprises…

Order of enqueuing is not a prediction of delivery. Your attitude should be one of “asynchronicity.”

Message Queueing Solutions

Advanced message-queueing protocol (AMQP) implementations. This is just one, formal class of messaging-queueing providers (but, the most common):

- Apache ActiveMQ
- RabbitMQ
- Amazon Simple Queueing Service
- Oracle Advanced Queueing (OracleAQ)
-
- Apache Apollo (ActiveMQ *citius*, *altius*, *fortius*)
- SwiftMQ
- Apache Qpid
- Windows Azure Service Bus
- ZeroMQ
- many more...

Solutions--get serious

Real options (in my humble opinion)

- Apache ActiveMQ (very old, very active open source community) ✓
- RabbitMQ (very smart Brits)

Non-options

- Apollo (over-kill, assumes bottlenecks not everyone has)
- Amazon Simple Queueing Service (Amazon AWS cloud)
- OracleAQ (evil and nasty--just say *No!*)

Message Queueing--Python

We need a STOMP client for use from Python. STOMP stands for *simple text-oriented messaging protocol*.

STOMP support is so clients can be written easily in C, Ruby, Perl, Python, PHP, ActionScript/Flash, Smalltalk to talk to ActiveMQ as well as any other popular message broker.

(Java and C# support is more *first-tier*. These are always provided by native libraries.)

Message Queueing--Python (continued)

Python example using STOMP. STOMP doesn't come absolutely for free:

```
~ $ python
Python 2.7.5 (default, Nov  3 2014, 14:26:24)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-7)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import stomp
Traceback (most recent call last):
  File "<stdin>", line 1, in
ImportError: No module named stomp
```

So, here's a STOMP client, the one Python seems to recognize, written by a Jason R. Briggs: stomp.py-4.0.13.tar.gz.* This version supports both Python 2.x and 3.x.

(*) https://pypi.python.org/pypi/stomp.py

Message Queueing--Python (continued)

Sending a message (i.e.: being a producer)

```
import time
import sys
import stomp

# Set up connection to (whatever queueing service supporting STOMP)...
conn = stomp.Connection()

# Start a session with the queueing service...
conn.start()
conn.connect()

# Now, put a message into the queue and sleep a bit...
conn.send( body=' '.join( sys.argv[ 1: ] ), destination='/queue/test' )
time.sleep( 2 )

# We're done!
conn.disconnect()
```

Message Queueing--Python (continued)

Receiving a message (i.e.: being a consumer)

```
import time
import sys
import stomp

class Listener( object ):
  # Post a listener to wait for an arriving message.
  def on_error( self, headers, message ):
    # What to do when an error receiving a message occurs
    print( 'received an error %s' % message )
  def on_message( self, headers, message ):
    # What to do when a message is received
    print( 'received a message %s' % message )

# Set up connection to (whatever queueing service supporting STOMP)...
conn = stomp.Connection()

# Set up a listener: what receives a message...
conn.set_listener( '', Listener() )

# Start a session with the queueing service...
conn.start()
conn.connect()

# Establish a subscription with the queueing service...
conn.subscribe( destination='/queue/test', id=1, ack='auto' )

# Now, sleep a bit...
time.sleep( 2 )

# We're done!
conn.disconnect()
```

Message Queueing--What else?

Once you’ve got Python STOMP, the rest is easy ‘cause folk have been breathing this air for years. There’s no lack of support via Google search, stackoverflow and other forums.

ActiveMQ (and most others, but not ZeroMQ) is...

- A formal software install on server hardware (RPM, Debian, Windows, Macintosh OSX, etc.).
- It runs as a service (got a *pid*, a memory footprint, etc.).
- You can look at what’s going on using any browser.