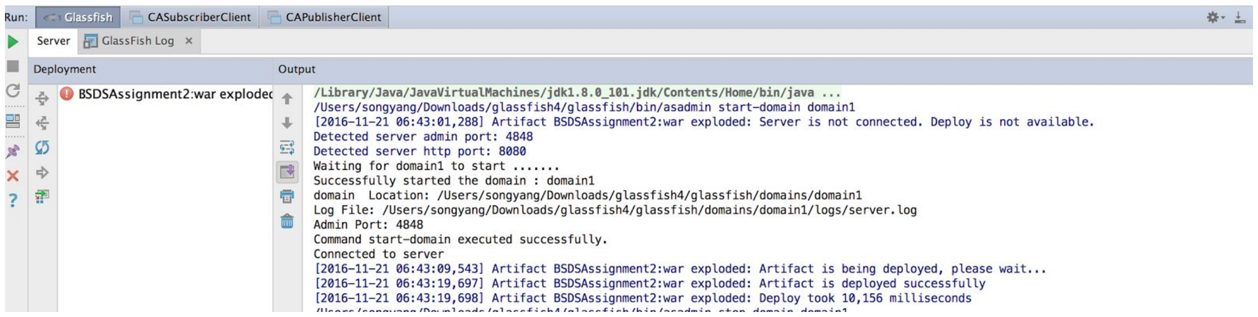
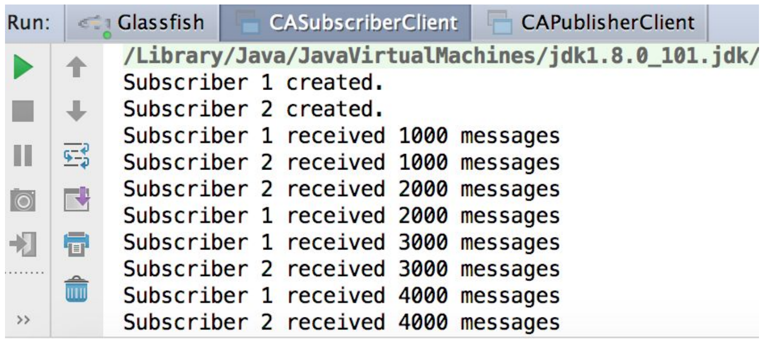
Step1: Basic Functional Correctness

First, start the glassfish server and deploy the web application. The output is:

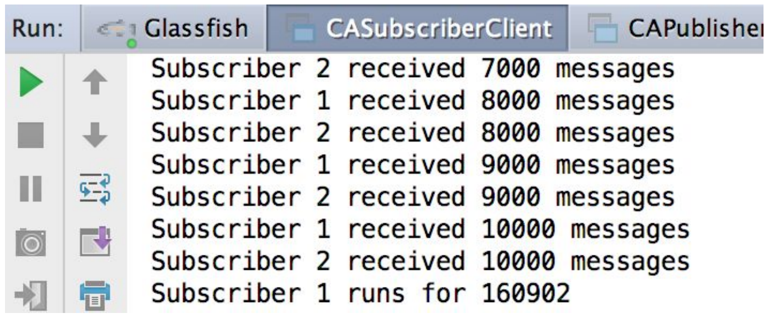


```
/Library/Java/JavaVirtualMachines/jdk1.8.0_101.jdk/Contents/Home/bin/java ...
/Users/songyang/Downloads/glassfish4/glassfish/bin/asadmin start-domain domain1
[2016-11-21 06:43:01,288] Artifact BSDSAssignment2:war exploded: Server is not connected. Deploy is not available.
Detected server admin port: 4848
Detected server http port: 8080
Waiting for domain1 to start .......
Successfully started the domain : domain1
domain  Location: /Users/songyang/Downloads/glassfish4/glassfish/domains/domain1
Log File: /Users/songyang/Downloads/glassfish4/glassfish/domains/domain1/logs/server.log
Admin Port: 4848
Command start-domain executed successfully.
Connected to server
[2016-11-21 06:43:09,543] Artifact BSDSAssignment2:war exploded: Artifact is being deployed, please wait...
[2016-11-21 06:43:19,697] Artifact BSDSAssignment2:war exploded: Artifact is deployed successfully
[2016-11-21 06:43:19,698] Artifact BSDSAssignment2:war exploded: Deploy took 10,156 milliseconds
```

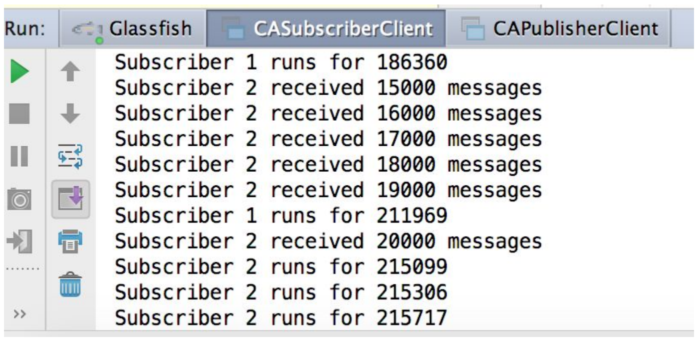
Then run two subscriber client threads. One subscriber subscribes to Topic1 and the other subscribes to Topic2. Below is the output (including running time) after the two subscriber received all the messages. The one who received 10000 messages ran for about 16 seconds, the other with 20000 messages ran for 21 seconds.



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_101.jdk/
Subscriber 1 created.
Subscriber 2 created.
Subscriber 1 received 1000 messages
Subscriber 2 received 1000 messages
Subscriber 2 received 2000 messages
Subscriber 1 received 2000 messages
Subscriber 1 received 3000 messages
Subscriber 2 received 3000 messages
Subscriber 1 received 4000 messages
Subscriber 2 received 4000 messages
```

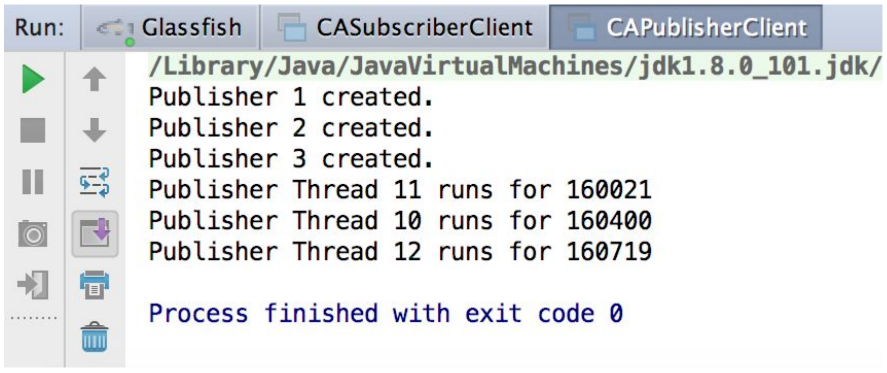


```
Subscriber 2 received 7000 messages
Subscriber 1 received 8000 messages
Subscriber 2 received 8000 messages
Subscriber 1 received 9000 messages
Subscriber 2 received 9000 messages
Subscriber 1 received 10000 messages
Subscriber 2 received 10000 messages
Subscriber 1 runs for 160902
```



```
Subscriber 1 runs for 186360
Subscriber 2 received 15000 messages
Subscriber 2 received 16000 messages
Subscriber 2 received 17000 messages
Subscriber 2 received 18000 messages
Subscriber 2 received 19000 messages
Subscriber 1 runs for 211969
Subscriber 2 received 20000 messages
Subscriber 2 runs for 215099
Subscriber 2 runs for 215306
Subscriber 2 runs for 215717
```

Finally start 3 publisher client threads. Two published to Topic1 and the other published to Topic2, each published 10000 messages. The running time for all 3 publisher threads is about 16 seconds (which is approximately the same as the running time of subscriber 1).

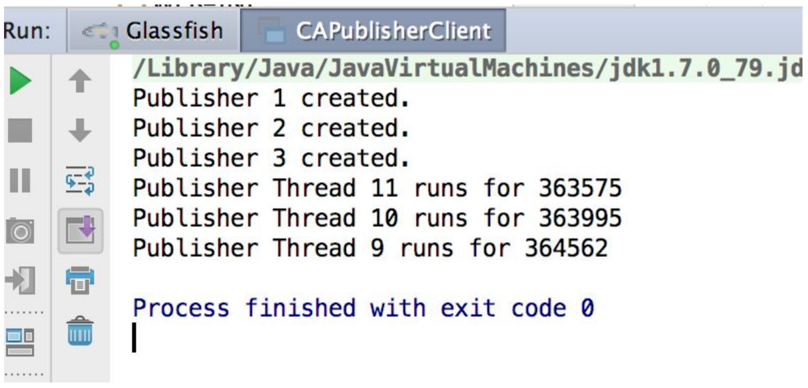


```
/Library/Java/JavaVirtualMachines/jdk1.8.0_101.jdk/
Publisher 1 created.
Publisher 2 created.
Publisher 3 created.
Publisher Thread 11 runs for 160021
Publisher Thread 10 runs for 160400
Publisher Thread 12 runs for 160719

Process finished with exit code 0
```

Step 2: Add Term Count

In this step I just ran publisher client first, since breaking down the message into words and store each word into database could be very time consuming. The published messages are words from Chapter 1 of A Tale of Two Cities, and each time each publisher will post the exact same content. The running result of publishers is:

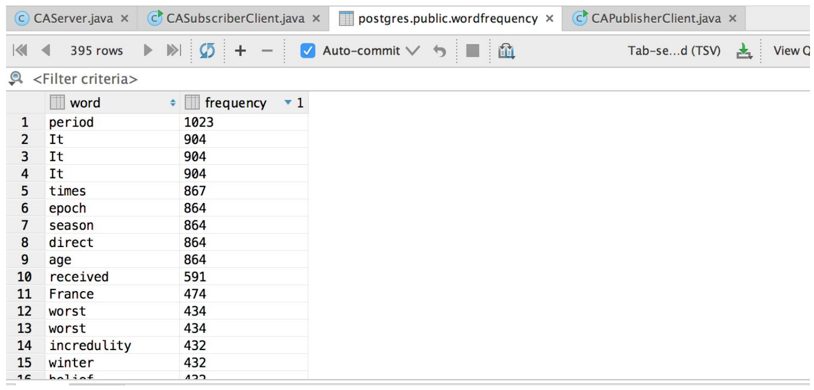


```
/Library/Java/JavaVirtualMachines/jdk1.7.0_79.jd
Publisher 1 created.
Publisher 2 created.
Publisher 3 created.
Publisher Thread 11 runs for 363575
Publisher Thread 10 runs for 363995
Publisher Thread 9 runs for 364562

Process finished with exit code 0
```

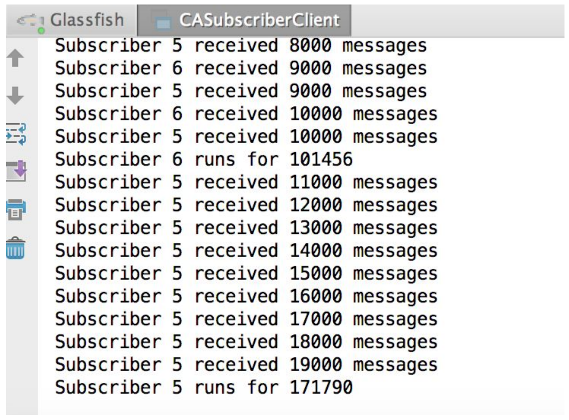
Running time of each publisher is about 36 seconds (without subscribers running at the same time). The wall time for publishers is a lot larger than that of step 1, mainly because of the read and write overhead to the Postgresql database.

The word count table of the database after publisher published all messages:



	word	frequency
1	period	1023
2	It	904
3	It	904
4	It	904
5	times	867
6	epoch	864
7	season	864
8	direct	864
9	age	864
10	received	591
11	France	474
12	worst	434
13	worst	434
14	incredulity	432
15	winter	432

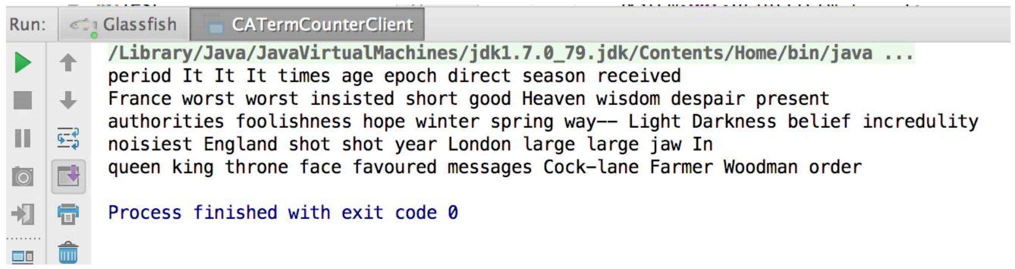
After publishers published all the messages, start the subscriber client and consume all the messages. Output:



```
Subscriber 5 received 8000 messages
Subscriber 6 received 9000 messages
Subscriber 5 received 9000 messages
Subscriber 6 received 10000 messages
Subscriber 5 received 10000 messages
Subscriber 6 runs for 101456
Subscriber 5 received 11000 messages
Subscriber 5 received 12000 messages
Subscriber 5 received 13000 messages
Subscriber 5 received 14000 messages
Subscriber 5 received 15000 messages
Subscriber 5 received 16000 messages
Subscriber 5 received 17000 messages
Subscriber 5 received 18000 messages
Subscriber 5 received 19000 messages
Subscriber 5 runs for 171790
```

Since there are no publishers running at the same time, the wall time of this step is shorter than step 1. The one with 10000 messages ran for about 10 seconds, while the other with 20000 messages ran for 17 seconds.

Then, call CATermCounterClient to get the top 50 most popular terms:

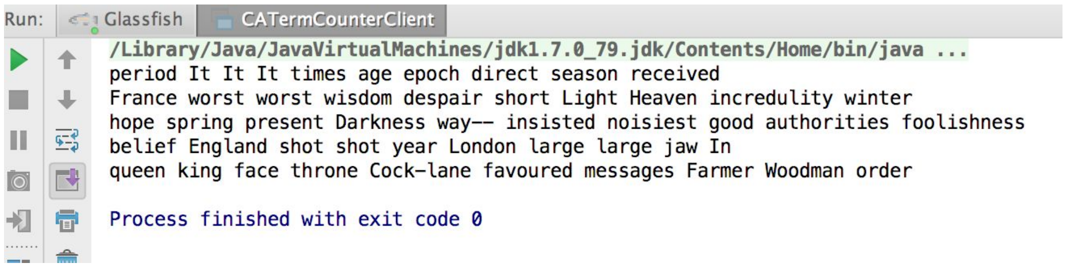


```
/Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/bin/java ...
period It It It times age epoch direct season received
France worst worst insisted short good Heaven wisdom despair present
authorities foolishness hope winter spring way-- Light Darkness belief incredulity
noisiest England shot shot year London large large jaw In
queen king throne face favoured messages Cock-lane Farmer Woodman order

Process finished with exit code 0
```

There seems to be some concurrent issue as there are duplicated terms in the result, but other than that this response corresponds to the records in the database.

Then ran publishers again, and call CATermCounterClient to get the top 50 most popular terms again:

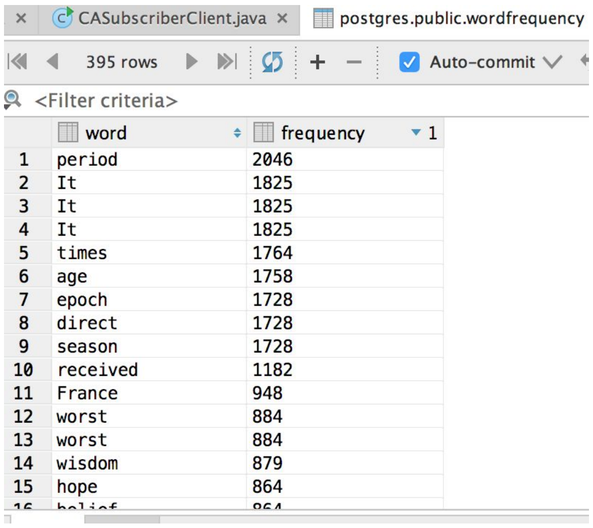


```
/Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/bin/java ...
period It It It times age epoch direct season received
France worst worst wisdom despair short Light Heaven incredulity winter
hope spring present Darkness way-- insisted noisiest good authorities foolishness
belief England shot shot year London large large jaw In
queen king face throne Cock-lane favoured messages Farmer Woodman order

Process finished with exit code 0
```

Most of the terms stay the same, but there are some minor changes. I think that could be due to some concurrent update issues on Postgresql. I didn't write any code on the server side to handle potential race conditions, instead I fully relied on Postgresql's concurrent management. I'll try to do some optimization on this in the following steps.

Let's look at the database records:



	word	frequency
1	period	2046
2	It	1825
3	It	1825
4	It	1825
5	times	1764
6	age	1758
7	epoch	1728
8	direct	1728
9	season	1728
10	received	1182
11	France	948
12	worst	884
13	worst	884
14	wisdom	879
15	hope	864
16	belief	864

Compare to the previous screenshot of database, most of the popular terms remain the same, and the counts double, which is as expected.