

Report:

Currently, I finished all the basic requirement of step1 and step2. However, from the test result, we see that the performance is really poor. From the file step1.output, unfortunately, I found that the wall time are really long and currently **not scalable** at all :(

- [Pub1] Totally cost 137627 ms
- [Pub2] Totally cost 189765 ms
- [Sub3] Totally cost 136237 ms
- [Sub4] Totally cost 180667 ms

There are several reasons that might be the cause:

- I used **synchronized methods rather than async methods** for all rest web service
- HTTP communication costs much time
- Json parser costs much time

The first one is supposed to be the most critical reason.

For step 2, I am using sqlite as the database backend. I can confirmed that the functionality is correct by reading the step2.output and the screenshot of result of getTopN method listed below. There're still some problems, where some of them are the same as described in step1:

- The problem of long wall time found in step1 still exist
- If there's no sync while manipulating database, then there's a high probability that will cause a segment fault (Yes it's a segment fault in JAVA). Once I add sync to each data CRUD method, the wall time becomes even longer though this problem does not exist anymore.
- I am trying to update each word occurred in a message to database after fully scanned a message. That means, if there're n different words in a message, then I need to run SQL query for n times to update the word count for each word. Since I am using SQL, it seems not an ideal way and I am trying to find a better way.
- Because database operations takes a lot more time, and the publishContent() method is called and handled in a synchronized order, the wall time of publisher increased a lot compared to step 1.
 - [Pub1] Totally cost 513368 ms
 - [Pub2] Totally cost 803858 ms, which is **more than 13 minutes!**
- For subscribers, there's not much change so the performance is similar to step1
- Because the processing time is too slow, there's even timeout happened (100ms) for subscribers. That's the reason why step2.output got 90k lines where step1.output runs the same test and only got 60k lines output. The extra 30k lines of log are mainly timeout info of subscribers.

First getTopN result after running clients once:

```
Message: 30000<br />yyy: 30000<br />xxx: 30000<br />Message0: 2<br />Message1: 2<br />Message2: 2<br />Message3: 2<br />Message4: 2<br />Message5: 2<br />Message6: 2<br />Message7: 2<br />Message8: 2<br />Message9: 2<br />Message10: 2<br />Message11: 2<br />Message12: 2<br />Message13: 2<br />Message14: 2<br />Message15: 2<br />Message16: 2<br />Message17: 2<br />Message18: 2<br />Message19: 2<br />Message20: 2<br />Message21: 2<br />Message22: 2<br />Message23: 2<br />Message24: 2<br />Message25: 2<br />Message26: 2<br />Message27: 2<br />Message28: 2<br />Message29: 2<br />Message30: 2<br />Message31: 2<br />Message32: 2<br />Message33: 2<br />Message34: 2<br />Message35: 2<br />Message36: 2<br />Message37: 2<br />Message38: 2<br />Message39: 2<br />Message40: 2<br />Message41: 2<br />Message42: 2<br />Message43: 2<br />Message44: 2<br />Message45: 2<br />Message46: 2<br />
```

Second getTopN result after running clients twice:

```
Message: 60000<br />yyy: 60000<br />xxx: 60000<br />Message0: 4<br />Message1: 4<br />Message2: 4<br />Message3: 4<br />Message4: 4<br />Message5: 4<br />Message6: 4<br />Message7: 4<br />Message8: 4<br />Message9: 4<br />Message10: 4<br />Message11: 4<br />Message12: 4<br />Message13: 4<br />Message14: 4<br />Message15: 4<br />Message16: 4<br />Message17: 4<br />Message18: 4<br />Message19: 4<br />Message20: 4<br />Message21: 4<br />Message22: 4<br />Message23: 4<br />Message24: 4<br />Message25: 4<br />Message26: 4<br />Message27: 4<br />Message28: 4<br />Message29: 4<br />Message30: 4<br />Message31: 4<br />Message32: 4<br />Message33: 4<br />Message34: 4<br />Message35: 4<br />Message36: 4<br />Message37: 4<br />Message38: 4<br />Message39: 4<br />Message40: 4<br />Message41: 4<br />Message42: 4<br />Message43: 4<br />Message44: 4<br />Message45: 4<br />Message46: 4<br />
```

Currently, the portion I am going to improve is to **make all methods asynchronized if possible**. I am a little busy these day abd that's the reason why it havn't been done yet. I am going to try **using JMS to buffer the income message to process the word count in an async mode** which means saving much time for publishers.