Publish and subscribe to system documentation

Based on Python

ywang531@hawk.iit.edu A20496705

一、 My machine information：

MacBook Pro (15-inch, 2018) - Intel UHD Graphics 630 1536 MB-16 GB

2400 MHz DDR4-2.2GHz six-core Intel Core i7

二、 Zip package structure：

According to the requirements, I have separated three folders: **Code**, **Docs** and **Misc**.

1. Code folder

Here is mainly the implementation code, the folder has the **Makefile** and **README.md** explain the requirements of the build environment and implement the common scripts of the publish subscription system.

Include those required in the assignment:

Publisher:

<PID> registerPublisher();

createTopic (PID, String topic);

deleteTopic (PID, String topic);

send (PID, String topic, String message);

Subscriber:

<SID> registerSubscriber ();

subscribe (SID, String topic);

List <String> pull (SID, String topic);

There are corresponding implementations

Also included are the required tests (performance tests, ping-pong tests, etc.) in this folder.

2. Docs folder

The main purpose is to store this document, including but not limited to evaluation reports, readme papers and documents

3. Misc folder

It is mainly to store some screenshot files, and the files running in the program will also be reflected here.

三、 Basic screenshot of operation

We will run step by step according to the instructions in README.md

1. Clean up the running environment and run initialization scripts

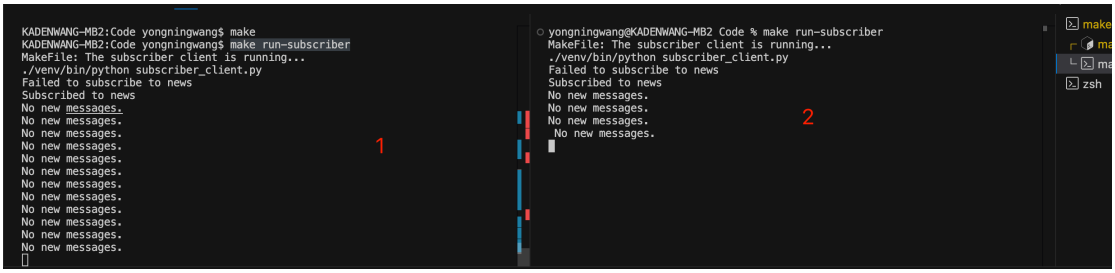
```
yongningwang@KADENWANG-MB2 Code % make setup
MakeFile: Creating a virtual environment...
python3 -m venv ./venv
./venv/bin/pip install -r requirements.txt
Collecting contourpy==1.1.1
  Using cached contourpy-1.1.1-cp39-cp39-macosx_10_9_x86_64.whl (247 kB)
Collecting cycler==0.12.1
  Using cached cycler-0.12.1-py3-none-any.whl (8.3 kB)
Collecting Cython==3.0.8
  Using cached Cython-3.0.8-cp39-cp39-macosx_10_9_x86_64.whl (3.1 MB)
Collecting fonttools==4.53.1
  Using cached fonttools-4.53.1-cp39-cp39-macosx_10_9_universal2.whl (2.8 MB)
Collecting importlib_resources==6.4.5
  Using cached importlib_resources-6.4.5-py3-none-any.whl (36 kB)
Collecting kiwisolver==1.4.7
  Using cached kiwisolver-1.4.7-cp39-cp39-macosx_10_9_x86_64.whl (65 kB)
Collecting matplotlib==3.7.5
  Using cached matplotlib-3.7.5-cp39-cp39-macosx_10_12_x86_64.whl (7.4 MB)
Collecting msgpack==1.0.7
```

2. Start the server program

```
yongningwang@KADENWANG-MB2 Code % make run-server
MakeFile: Running server...
./venv/bin/python server.py
Server listening on localhost:12345
```

3. Activate any number of subscribers to subscribe to the news topic

   (Need to open an additional shell session)



4. Launch a publisher and automatically send a Hello Word message

   in the news topic when launched

```
yongningwang@KADENWANG-MB2 Code % make run-publisher
MakeFile: The publisher client is running...
./venv/bin/python publisher_client.py
```

5. **If you want to publish messages multiple times, activate the publisher multiple times**

```
yongningwang@KADENWANG-MB2 Code % make run-publisher
MakeFile: The publisher client is running...
./venv/bin/python publisher_client.py
yongningwang@KADENWANG-MB2 Code % make run-publisher
MakeFile: The publisher client is running...
./venv/bin/python publisher_client.py
yongningwang@KADENWANG-MB2 Code % make run-publisher
MakeFile: The publisher client is running...
./venv/bin/python publisher_client.py
yongningwang@KADENWANG-MB2 Code % make run-publisher
MakeFile: The publisher client is running...
./venv/bin/python publisher_client.py
yongningwang@KADENWANG-MB2 Code % make run-publisher
MakeFile: The publisher client is running...
./venv/bin/python publisher_client.py
yongningwang@KADENWANG-MB2 Code % make run-publisher
MakeFile: The publisher client is running...
./venv/bin/python publisher_client.py
yongningwang@KADENWANG-MB2 Code %
```

6. **Observe whether subscribers can receive messages**

```
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
Received messages: ['Hello, world', 'Hello, world', 'Hello, world']
Received messages: ['Hello, world']
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
```

```
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
Received messages: ['Hello, world']
Received messages: ['Hello, world', 'Hello, world', 'Hello, world']
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
No new messages.
```

Above, we have completed the basic debugging of the publish subscribe server.

We are based on multi-threaded implementation, where different subscribers only receive their own messages without interfering with each other, and their own subscriber message buffer pool will be cleared after receiving the message. It is worth noting that there is no requirement for the startup

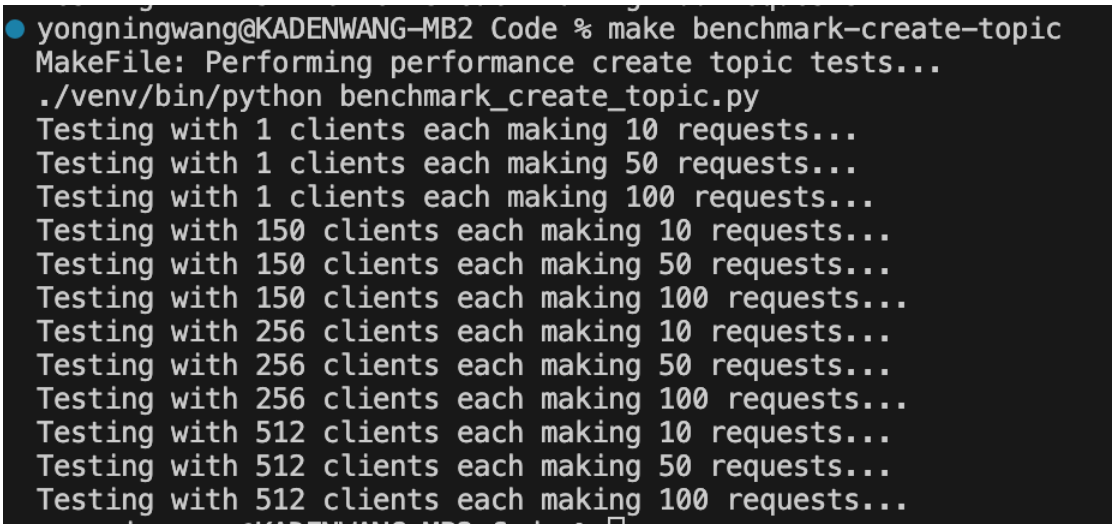
sequence from step 3 to step 5.

- If the publisher has started but the subscriber has not started. It will pull up historical topic messages when the subscriber starts
- If the subscriber starts but the publisher does not, the subscriber will continuously ask the service for any new messages every 1 second
- If the publisher quickly publishes many messages, subscribers will pull all the messages within a specified time interval in one polling (simulating high concurrency)

四、 Performance testing and analysis

1. Create_topic interface testing

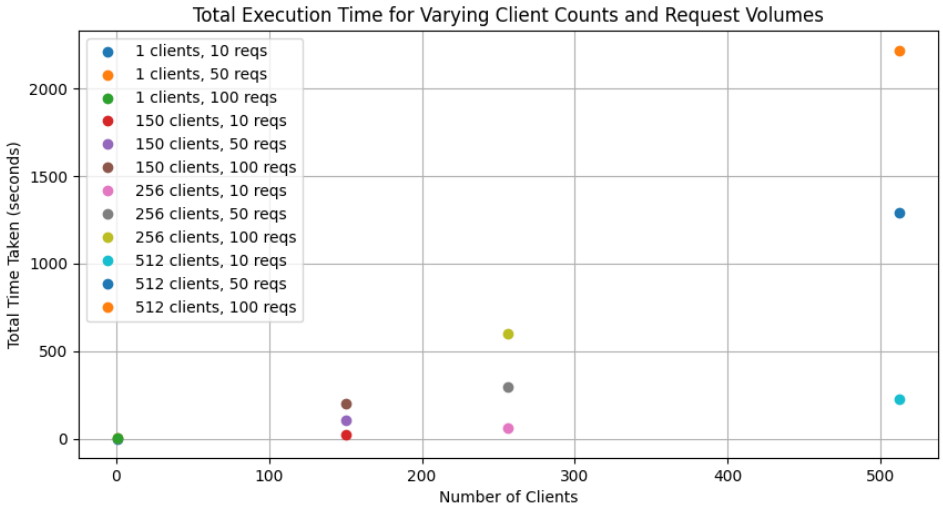
1.1 Running screenshot:



```
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
Sending response: {'status': 'Topic created'}
```

1.2 Comparison method

Start the server first and run benchmark create topic on my computer to observe. Compare the performance differences of launching different clients and sending different numbers of requests for each client. The following picture shows the running result of my computer:



1.3 analyze

Observation method: Mainly look at the error rate and time.

In our test case, because the publishing and subscription system is relatively simple, and a lot of exception processing is done, the error rate is not

high at present, mainly depending on the time. Looking at the figure, we can observe that:

- Low latency at low client count: When the number of clients is small (such as one client), the execution time is relatively low regardless of changes in request volume. This indicates that the server responds quickly at low loads.

- High latency with high number of clients: As the number of clients increases (especially up to 512 clients), execution time significantly increases, especially when the request volume per client is also high (such as 100 requests per client).

The goal is to determine how to handle as many requests as possible without sacrificing response speed, and to find a balance between a small number of clients and a large number of requests per client. For example, a configuration of 150 clients making 100 requests each might be a better choice because it handles a large number of requests while maintaining a low overall execution time.

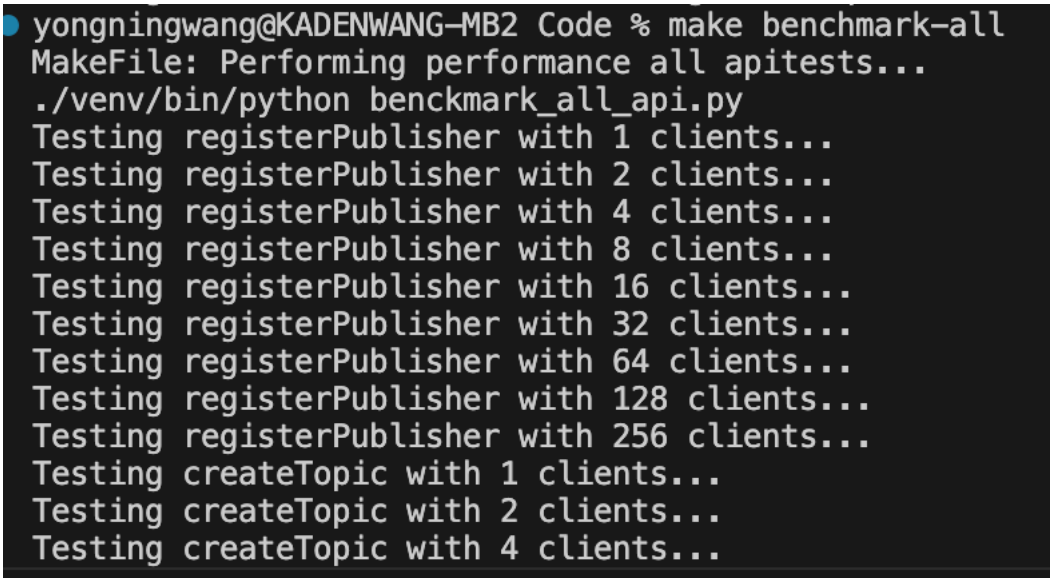
1.4 Customized testing

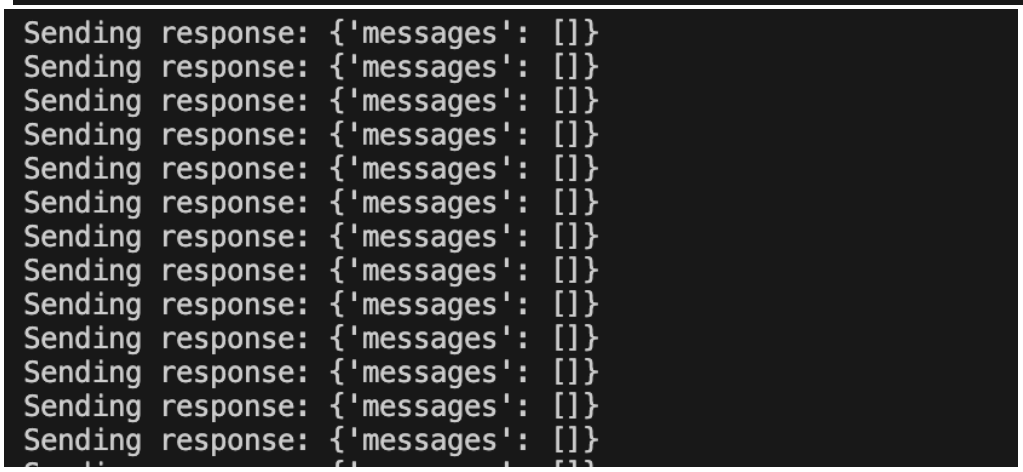
If you want to test the maximum throughput of your machine, you can change the maximum number of clients and message concurrency here.

```
if __name__ == "__main__":
    client_counts = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
    num_requests_per_client = [10, 50, 100]
    all_results = {}
```

2. other interface testing

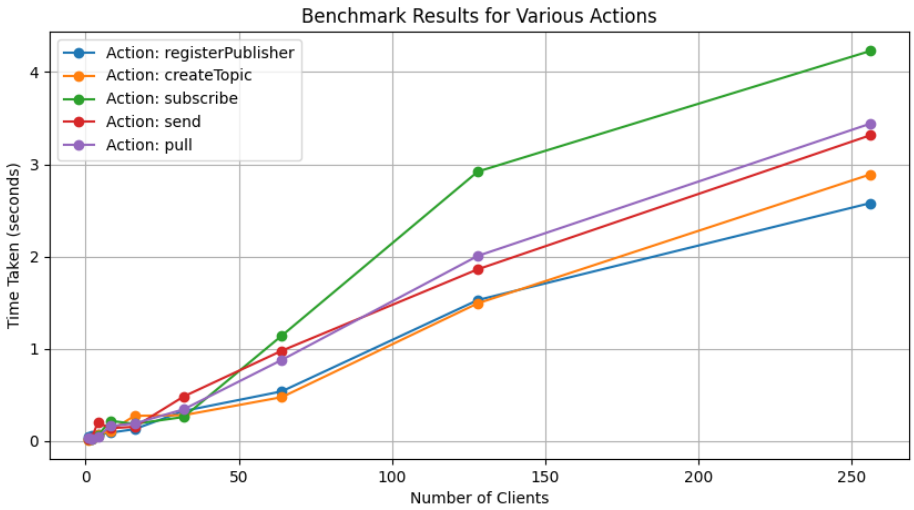
2.1 Running screenshot:





2.2 Comparison method

Similar to the tests for the create topic interface, we compared the performance of building different client counts with 100 concurrent counts.



2.3 analyze

The chart shows the execution time of different API operations (registering Publisher, creating Topic, subscribe, send, pull) at different numbers of clients. Based on this chart, we will conduct some key analysis and propose optimization suggestions.

Performance varies with increasing number of clients: the execution time of all operations increases with increasing number of clients. This indicates that as the system concurrency increases, the processing capacity of the server is challenged.

Performance differences of different operations:

- The slow increase in execution time for registering **Publisher and creating Topic** may indicate that these operations are relatively lightweight and require less resources.

- The execution time of **subscribe and send** increases rapidly, especially when there are a large number of clients, which may indicate that these

operations are more resource intensive or involve more complex logic.

- The execution time of the **pull** operation initially increases slowly, but then significantly increases, which may indicate a performance bottleneck when processing large amounts of data or message pulls.

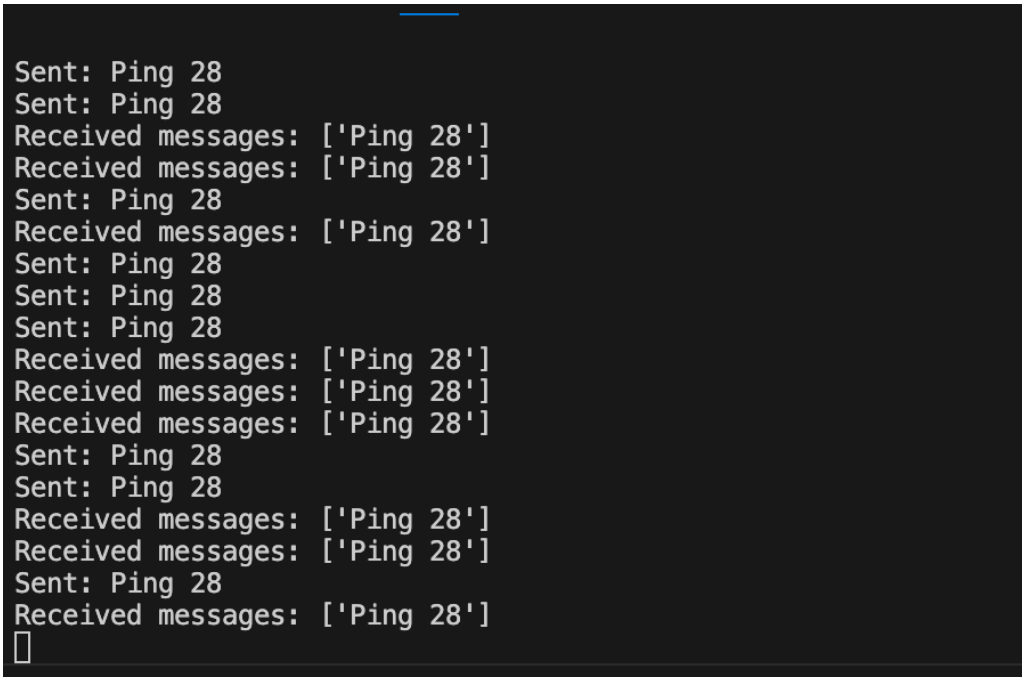
2.4 Customized testing

If you want to test the maximum throughput of your machine, you can change the maximum number of clients here.

```
# main
if __name__ == "__main__":
    client_counts = [1, 2, 4, 8, 16, 32, 64, 128, 256]
    num_requests_per_client = 100
    # all need to be test action of lists
    actions = ["registerPublisher", "createTopic", "subscribe", "send", "pull"]
    all_results = {action: {} for action in actions}
```

3. Ping-pong testing

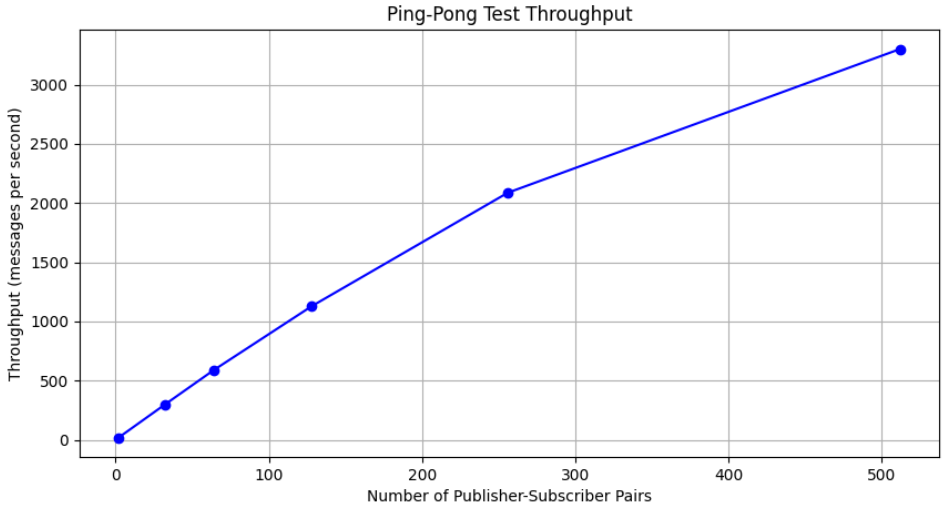
3.1 Running screenshot:



```
Sending response: {'messages': ['Ping 49']}
Sending response: {'messages': ['Ping 49']}
Sending response: {'status': 'Message sent: Ping 49'}
Sending response: {'status': 'Message sent: Ping 49'}
Sending response: {'status': 'Message sent: Ping 49'}
Sending response: {'status': 'Message sent: Ping 49'}
Sending response: {'status': 'Message sent: Ping 49'}
Sending response: {'status': 'Message sent: Ping 49'}
Sending response: {'messages': ['Ping 49']}
Sending response: {'messages': ['Ping 49']}
Sending response: {'messages': ['Ping 49']}
Sending response: {'messages': ['Ping 49']}
Sending response: {'messages': ['Ping 49']}
Sending response: {'messages': ['Ping 49']}
Sending response: {'status': 'Message sent: Ping 49'}
Sending response: {'status': 'Message sent: Ping 49'}
Sending response: {'messages': ['Ping 49']}
Sending response: {'messages': ['Ping 49']}
```

3.2 Comparison method

Adjust the number of publisher subscriber pairs through fixed message concurrency, observe changes in time consumption and performance.



3.3 Analyze

The chart shows that as the number of publisher subscriber pairs increases, throughput (messages per second) exhibits an approximately linear growth. This indicates that the system can effectively scale to handle more concurrent communication pairs without significant

performance bottlenecks or degradation.

The throughput increases linearly with logarithm, indicating that the system architecture has good scalability. When adding more publishers and subscribers, the system can effectively handle the increased load, which is an important indicator in distributed system design.

3.4 Customized testing

If you want to test the maximum publisher - subscriber of your machine, you can change the maximum number of clients here.