CS 6381: Distributed Systems Principles: Spring 2023

Programming Assignment 1: Centralized Discovery, Multi-Dissemination Strategy Publish-Subscribe

Handed out: 01/18/2023; **Due dates**: Milestone 1: 1/27/23 11:59 pm; Milestone 2: 02/03/23 11:59 pm; Milestone 3: 02/10/23 11:59 pm

Note 1: Programming Assignments are to be done individually. Discussions on Slack are highly encouraged. Any help for those who have not done the Cloud or Networking course and are stuck deploying the pods or running Mininet, etc is also allowed. But no direct help in writing code is permitted. All work for the specifics of the assignment must be an individual effort.

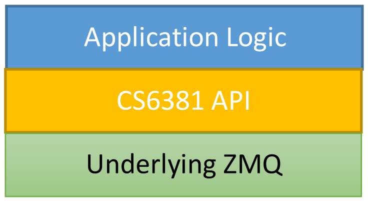
Note 2: Please read the writeup carefully and in its entirety, and ask clarifications, if any.

Note 3: To use ZMQ, you will need to install the pyzmq Python package. Since we will be using Python3, install it via pip3

Overview

In this assignment we will build upon the PUB/SUB model supported by the ZeroMQ (ZMQ) middleware. ZMQ is a modern messaging middleware that provides the necessary building blocks and a range of patterns to create a variety of distributed systems. However, one of the downsides of ZMQ's approach is that there is no anonymity between publishers and subscribers. A subscriber needs to know where the publisher is (i.e., subscriber must explicitly connect to a publisher using its IP address and port). So, we lose some degree of decoupling with such an approach (recall that the definition for an ideal publish-subscribe calls for time, space, and synchronization decoupling). A more desirable solution therefore is where the application logic of the publishers and subscribers remains anonymous to each other; naturally somebody and at some level must still need to maintain these associations thereby breaking the definition but that is ok as long as the application-level logic complies with the ideal definition. So, in our assignment, we are going to push this responsibility to a layer of middleware that we are going to design, and then have our application logic use the API of our middleware instead of directly using ZMQ API.

Accordingly, in this assignment, we will build a small additional layer of pub/sub middleware on top of existing ZMQ capabilities to support anonymity. See the diagram below of the expected abstraction. The Yellowish colored layer shown in this diagram is that layer of middleware that we are going to build. We will then test our application logic by using the API of our middleware layer.



A starter code is being provided in this directory to realize this vision. The Publisher part is more or less complete but the design of the remainder of the entities is to be completed by students.

Feel free to use whichever underlying capabilities/patterns provided by ZMQ inside your middleware layer that you feel appropriate. For instance, the REQ-REP and PUB-SUB are the most likely patterns of ZMQ you will need (and maybe XPUB/XSUB). No matter which features of ZMQ you use in the middleware layer, the direct use of ZMQ operations should be hidden from your publisher and subscriber application logic. The application code should use only the APIs of your middleware, which then translate into ZMQ calls under the hood and are hidden from application logic. See starter code.

Specification

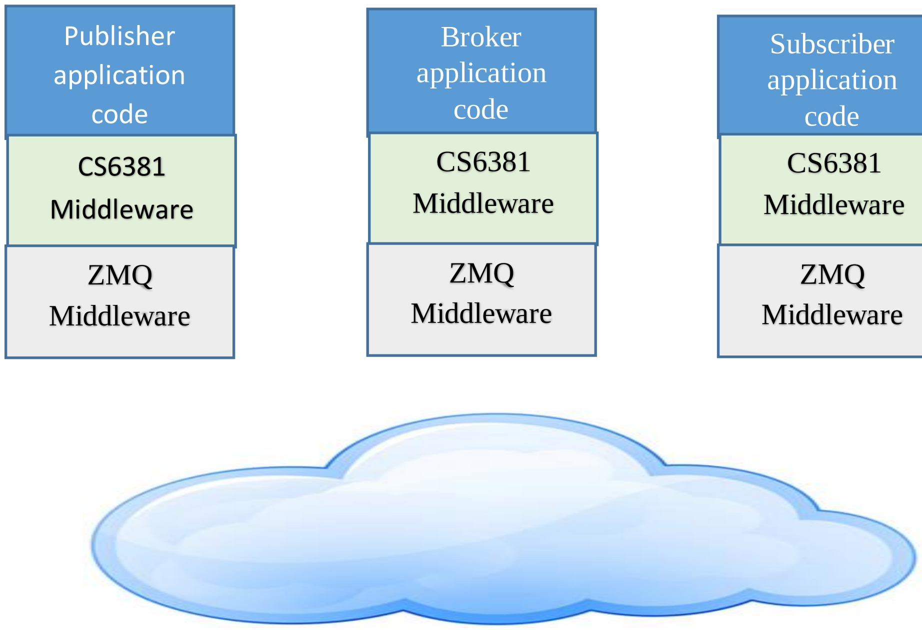
The specification comprises two independent parts:

(a) **Discovery/Lookup strategy:** In this assignment, we will maintain a single, centralized repository/registry that serves as the well-known, go-to place for discovery/lookup by all the publishers and subscribers in our system.

(b) **Information Dissemination strategy:** Here, we are to support two mutually exclusive and system-wide configurable strategies to disseminate information:
   a. **Direct:** where a publisher directly sends information to the subscriber (after the matching is done)
   b. **ViaBroker:** where a centralized, single broker is used by all publishers to disseminate information to interested subscribers.

Write your code in such a way that the approach is configurable consistently across the system to use one of the two dissemination approaches when the system is deployed. For example, one might use the Python configparser class to capture such common configurations. In other words, if the "Direct" approach is chosen, then all of publishers, broker and subscribers must be configured to use that strategy; you cannot mix things else the system will not work properly as per the specifications. Both approaches should be supported in your code.

Ensure that your code is designed with extensibility in mind because subsequent assignments are going to enhance this basic design by adding various other criteria. A general architecture is shown in the figure below where we show the architectural elements of our assignment. **We will use Python3 for our assignments.**



After writing the code for the assignment, you should be able to conduct a variety of end-to-end performance measurement experiments under a variety of load scenarios to get a sense of the impact on performance due to amount of data sent, latency for dissemination for the two different strategies and as the number of publishers/subscribers increase as well as topics get published, pinpointing the source of bottlenecks, etc in each of the dissemination approaches. Automation in this testing process is preferred so that you do not have to manually set up each experiment. I suggest that in the topic message that you send from publisher to subscriber, you also include the timestamps so that a time-series information can be stored in something like InfluxDB and then you can run variety of analytics on the collected data. All of this information can be serialized using Protobuf. See starter code for example.

Assumptions Made

In this assignment we make the following assumptions:

- There are no failures, i.e., no component fails.
- There are no late joining publishers or subscribers. Nobody leaves once they have arrived.
- The system is static i.e., everything is specified ahead of time. In other words, all publishers and subscribers are known ahead of time, i.e., everyone is present in the system along with the topics they publish (for publishers) or topics they are interested in (for subscribers) before the actual dissemination starts.

API Specification and Starter Code

A starter code with sample API is provided as a guideline in the ProgrammingAssignments folder on Box. It provides a minimal common API that everyone can use to build upon. Message types used by the Discovery service are define in the protobuf *.proto file. Starter code will be explained in class.

Milestones and Rubrics

The assignment will be developed in three phases (called milestones). The three milestones and minimum expected requirements that serve as the rubrics are as follows:

Milestone1: Points 30

- Develop the logic for Subscriber (Appln and MW) and Discovery (Appln and MW)
- Centralized registry with registration by publishers and subscribers
- Testing of code on Mininet network topologies on laptop VM
- At a minimum, both publishers and subscribers must be able to register with the Discovery service. The Discovery service should reply with a "true" once the system is ready.
- Optional: Make video of the operation and upload

Milestone 2: Points 30

- Everything from milestone 1
- Dissemination strategy using the direct approach
- Publishers should be able to disseminate; subscribers should be able to receive from every publisher who publishes the topic they are interested in.
- Testing of code on Mininet network topologies on laptop VM
- Test for all the different performance measurements under different load conditions for the Direct dissemination approach.
- Optional: Make video of the operation and upload

Milestone 3: Points 40

- Everything from milestones 1 and 2. Now support the ViaBroker approach => Broker logic must be developed.
- The system should be configurable to use either of the dissemination strategies system wide.
- Should work with all the properties mentioned in the specification of the assignment.
- Should work with multiple publishers publishing multiple different publications and multiple subscribers, all of them distributed over different hosts over different kinds of network topologies that we can create in Mininet and in cloud-based deployment (if we get this working in time else only Mininet).
- Use the InfluxDB ecosystem to save all the time series results of experimentation
- Do end-to-end measurements comprising the time taken between publication and receipt of information per publisher-subscriber pair. Since the clock is the same on all emulated mininet hosts, we do not have the issue of clocks drifting apart from each other
- Plot graphs comparing the two approaches under different load scenarios.
- Make a video of the entire assignment working with all milestones to make it easy for the grader to grade.

Code Availability and Grading

Ideally, using a personal github and private repository (so that the code is not publicly available) will be a good way to do the development. We will utilize peer grading for the 2U online MS section while our TA will grade the assignments for on-ground students; the peer grading responsibilities will be provided in class and in Box. You may need to invite your peer after your milestone is done to evaluate your code but preferably if there is a video, it is easiest for the grader.

Please refer to the generic programming assignment grading guidelines for the grading approach and rubrics.

Feedback to Instructor (for 2U online section)

The peer grader should send a detailed report to the instructor on how the submitted assignment met the grading requirements and grade received by the submitting peer along the specified rubrics the grader used.