# **CS 6381: Distributed Systems Principles, Spring 2023**

**Programming Assignment 4: Fault-Tolerant Publish-Subscribe Using ZMQ and ZooKeeper With Load Balancing and QoS Properties**

**Handed out:** 03/31/2023; **Due dates**: Milestone 1: 04/08/23 11:59 pm; Milestone 2: 04/15/23 11:59 pm; Milestone 3: 04/22/23 11:59 pm

**Note 1:** Programming Assignments are to be done individually. Discussions on Slack are highly encouraged. 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. Please do not wait till the last day.

**Note 3:** Use the progassign4 channel in Slack for discussions on this Assignment. Also, please try your level best to check if your question was already asked and answered. Please monitor the different threads.

**Overview**

In this assignment we will reuse all the code from assignment3. Moreover, we will demonstrate the approach for Broker-based communication only. We make the following three enhancements to the code base from PA3:

1. **Load balancing, in addition to fault tolerance:** In Assignment 3, we introduced broker fault tolerance using primary-backup scheme. But we did not care about the load on the primary broker. In this assignment we will introduce load balancing, in that there could be multiple primary replicas of the broker where a copy or copies of the primary broker are created when the load on one or more of the primary brokers (particularly in approach #2 where all communication is thru the broker) goes above a user-defined threshold. After a new load balanced primary replica is formed, the existing load should be rebalanced among the new primary replicas. Load balancing decisions can be made based on topic type or number of pubs/subs per topic or a combination of these (this is your choice). As before, fault tolerance should be maintained but now in addition we need load balancing. Any failure of a load balanced primary replica should ensure that one of the backups is promoted to such a load balanced primary replica. You can keep a single pool of backups or can partition the replicas if you decide to do so (your choice). When the load on the system falls below a threshold, one such load-balancing primary replica should stop handling any load, demoted to a backup, and all its load redistributed to remaining load-balanced primary replicas. Simplest way that I can think of (unless there is a better suggestion) to accomplish this without disrupting existing logic is to have a separate partitioned group of replicas designated to take on the role of a primary load balanced replica.
2. **Ownership strength:** Recall that up until now, a subscriber would end up receiving data from all publishers that published on the topic that that subscriber was interested in. However, now we will introduce the concept of ownership strength. Publishers publishing the same topic will maintain an ownership strength. You could use ZooKeeper as a way to assign ownership strength values, i.e., just like you do leader election, i.e., whoever comes first gets the highest ownership strength, and so on. You could start assigning monotonically increasing numbers for ownership strength where number 1 is highest strength and so on. NOTE that ownership strength is tied to a topic and not a publisher. For instance, if a publisher P is publishing two topics, T1 and T2, it is possible that it may have highest ownership strength for T1 but not for T2.

*How does ownership strength work?* In prior assignments, if you had more than one publisher publishing the same topic, all the events from all those publishers would get relayed to the interested subscribers. With ownership strength, information from only the highest ownership strength publisher gets relayed to the subscriber. The question is whether the publishers coordinate among themselves (via ZooKeeper) so that only the highest ownership publisher sends, or whether all publishers send info to the subscriber (via the broker) and its middleware logic will discard messages from non-highest ownership strength publishers (or in the broker-based approach, the broker can do the same and stop unwanted messages from going to subscribers).

Since publishers may die, if a publisher with highest ownership strength for a given topic were to die, the publisher with the next highest will assume ownership and start disseminating the information.

1. **History:** Recall that up until now, any late joining subscriber will lose information that is already published by a matching publisher. To handle this problem, we allow publishers to maintain a finite history of most recent N number of samples published, where N is configurable. Thus, for history QoS, we will require that the last N samples of information published on a topic be preserved in a sliding window fashion. N is a parameter that you supply. Note that you will have to comply with the “offered vs. requested” model and the dominance relationship. In other words, if a publisher maintains last 10 samples while a subscriber wants last 20 samples, then this subscriber cannot get the data even if the topic of interest is common. Thus, the matching rules need to be modified. Just like ownership strength, history is also associated with a topic. Thus, it is possible that a publisher publishing on topics T1 and T2 could maintain 10 and 20 samples, respectively, for the two different topics it publishes.

Everything else is the same as before and the experiments are the same.

**Assignment-Specific Demonstration and Rubrics**

Perform experiments for the Broker-based approach only involving both load balancing and fault tolerance. Show performance comparison with and without load balancing. It is highly recommended to do this assignment on the Cloud due to the larger scale of the assignment, which may be hard to reproduce on Mininet.

**Milestone 1 (ungraded):** Load balancing should work with everything that you had in Assignment #3.

**Milestone 2 (ungraded):** Introduce Ownership Strength to the above.

**Milestone 3 (100 points):** Introduce History QoS combined with Ownership Strength. This will impact the matching criteria based on offered vs requested model and using dominance relationship.

Please refer to the generic programming assignment grading guidelines for the grading rubrics. Thus, ensure that your assignment fulfills all the requirements as specified and meets the rubrics requirements including the README.

**Feedback to Instructor**

For the online class, the peer grader should send a detailed report to the instructor (CCing the student) on how the submitted assignment met the grading requirements and grade received along the specified rubrics they used.