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

**Programming Assignment 3: Fault-Tolerant Publish-Subscribe Using ZMQ and ZooKeeper**

**Handed out:** 03/03/2023; **Due dates**: Milestone 1: 03/11/23 11:59 pm; Milestone 2: 03/25/23 11:59 pm; Milestone 3: 04/01/23 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 progassign3 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.

**Note 4:** Since I had to make significant changes in last year’s writeup and overwrite the existing file, it is possible that some references to old material may still be present. Also, some requirements may not have been completely specified and/or may sound ambiguous. Please report these and I will immediately fix things in this writeup.

# **Overview**

Since all the assignments have the same final objective, we do not repeat the description but focus only on the changes. In Assignment 1, we focused on centralized discovery and a multi-dissemination approach. In assignment 2, we focused on a load balanced discovery service by virtue of using the Chord DHT algorithm. In this assignment, we will not change the basic APIs and strategies for discovery and dissemination. However, we will be introducing fault tolerance. I was also thinking of adding Load Balancing to brokers but I will push that requirement to PA4. Also, due to the major difficulties we faced in implementing the DHT, we will revert to centralized Discovery. So, you should be able to reuse all of your code for that part from Assignment #1 but with the new changes described below.

**Key changes from Assignments #1 and #2:**

* We will provide warm-passive fault tolerance capabilities to both Discovery and Brokers.
* Apache ZooKeeper will be used as the coordination service.
* Publishers and subscribers can arrive and leave at any time. So, they must perform both register and deregister when they gracefully enter/exit
* There is no “is\_ready” check anymore, however, publishers and subscribers cannot do anything unless they have registered successfully with the primary replica of the Discovery service and know who their broker is in the ViaBroker dissemination strategy
* On failure, publishers/subscribers/broker must know the new Discovery primary replica (possibly via Watch mechanism)
* We will introduce Kubernetes-based testing of the code apart from Mininet, and I will teach how to use this in class
* You will need to write effective experiment scripts that can showcase dynamic fault injection and dynamic arrival/departures of publishers/subscribers
* To keep our focus on the fault tolerance objectives, although Direct dissemination will work without any problems, we will focus more on the Broker-based approach as even our broker can fail.

# **Assumptions Made**

In this assignment we make the following assumptions:

* Only one instance of the ZooKeeper service executes, and it never fails.
* Both Discovery and Broker use warm-passive replication.
* Publishers and Subscribers can come and go at any time.
* Although we have restricted ourselves to 9 topics, we work with the assumption that the topics are also not known ahead of time.

# **Specification**

Assignment 3 focuses on fault tolerance of Discovery and Broker. We are going to address several of the requirements like fault detection, primary election, registration and lookups, barrier needed to maintain replica group quorum, and whatever other coordination capabilities we need through the effective use of coordination mechanisms and recipes provided by the Apache ZooKeeper framework. You may need to design your own recipes as needed. The key new requirements for this assignment are as follows:

* Run a single instance of the ZooKeeper service that everyone knows how to get to (so this is like our centralized discovery from Assignment #1)
* Every entity in our system will somehow be known to the ZooKeeper service and maintained in the metadata tree somewhere. See the set/get operations from the ZooKeeper scaffolding code.
* Run multiple centralized Discovery instances and let them elect a primary. However, we will need to build an “adapter” logic inside our Discovery service because its publisher/subscriber-facing API is different from the ZooKeeper-facing API provided by Kazoo. Thus, internally inside the Discovery, you should be able to translate between the two APIs.
* To maximize reuse of code, we will let publishers/subscribers/broker to use original API to talk to Discovery service, which in turn translates messages to and from ZooKeeper.
* Any publishers/subscribers arriving before a primary of Discovery is elected should not be able to do any dissemination.
* In the ViaBroker dissemination approach, we will run multiple brokers also in a warm-passive replication setup with some broker chosen as the leader via the ZooKeeper recipes
* Publishers, Subscribers, Brokers and Registries can die unexpectedly (e.g., pressing a control-C) instead of the graceful exit. You can script this behavior by sending a Kill Signal to the running process from your experiment coordinator process/script.
* It is possible that during recovery, some samples published by the publishers are lost because the new primary broker is not yet available. Understand these implications. These are unavoidable.

**Helpful resources:**

* Please see the DEBS2017 Paper that we had published and which is kept in this same directory. It shows how we had used ZooKeeper and what kind of metadata tree we had created for our needs. This is just a guideline, as it is likely that your ZooKeeper metadata tree will have a structure like this with some modifications.
* Please see the ZooKeeper scaffolding code example
* Slack Q&A in progassign3 channel
* Office hour (on request)

**Note:** In prior implementations of ZooKeeper, the watch mechanism had to be re-set by the application every time it receives a notification on that watch and wants to continue watching after that. But this behavior appears to have changed. Please confirm this. Also, understand the implication of permanent and ephemeral nodes. You will need to take advantage of their qualities for coordination purposes.

**Package Installation**

You will need the following additional packages for this assignment beyond what you already have installed.

* Latest stable release of Apache ZooKeeper. At this time version 3.7.1 is the stable version. It will download the tar.gz file. You will need to uncompress (use gunzip) and untar (use tar command). I use a shortcut like this “zcat <name of the tar.gz file> | tar xpof - ”. Note that | is the pipe operator of the shell. The hyphen after xpof is needed.
* Java 11. Best way to do this is “sudo apt install default-jdk" on Ubuntu.
* Python binding to ZooKeeper provided by the kazoo package. Do this as appropriate depending on systemwide or virtualenv installation that you are using.

# **API Specification**

The API does not really change much from what you had in Assignment #1 and #2. If you need additional capabilities, feel free to add them to your design. However, Discovery to ZooKeeper adapter will be needed.

# **Starter Code**

No specific starter code has been developed so far as we already have all the logic ready for publishers, subscribers, brokers and centralized discovery. Moreover, scaffolding code exists showing how to use the Apache ZooKeeper through the Kazoo Python binding.

# **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:

**Milestone 1 (ungraded but submit status update)**

* ZooKeeper service is deployed and is accessible.
* Adapter logic to go between existing centralized Discovery and Kazoo client, and ability of this composite to communicate with ZooKeeper.
* Registration and de-registration of publisher/subscribers with the single instance of centralized discovery

**Milestone 2 (ungraded but submit status update)**

* Broker leader election both first time and in the event of broker failure.
* All watch mechanisms as specified including informing publishers/subscribers of primary broker.
* The viaBroker strategy from Assignment 2 should now work using the primary broker as elected using ZooKeeper

**Milestone 3 (100 points)**

* All remaining cases including both graceful and unexpected termination of publishers and subscribers.
* As before anyone can come at any time.
* Do end-to-end measurements (time between publication and receipt of information; since the clock is the same on all emulated hosts, we do not have the issue of clocks drifting apart from each other). Plot graphs and have them saved in the repo that you maintain.
* Compare these performance results to the ones you got from Assignment #2 to see what impact, if any, is observed on performance due to recovery and coordination mechanism.
* Video and github access to your peer grader.

# **Code Availability and Grading**

As in previous assignments, it will be good to have a Github private repo where you are maintaining your code. For the online section, you may need to invite the grading peer in case the video is insufficient. For on-ground, the same holds if our TA Ziran is unable to grade the assignment based just on the video. For on-ground section, I am going to try setup a “github classroom” which might make it easier for Ziran to grade things and for you to share the code with her.

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

**Feedback to Instructor**

The peer grader in the online section 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 that the grader used. Please CC the student whose assignment you have graded. The TA will also provide me an overall idea of how students did in the on-ground class.