Distributed Systems
CS60002

Implementing a Distributed Queue

Date: Jan 31, 2023

Contributed By :

Esha Manideep Dinne	-	19CS10030
Amartya Mandal	-	19CS10009
Divyansh Bhatia	-	19CS10027
Anish Sofat	-	19CS10011
Rupinder Goyal	-	19CS10050

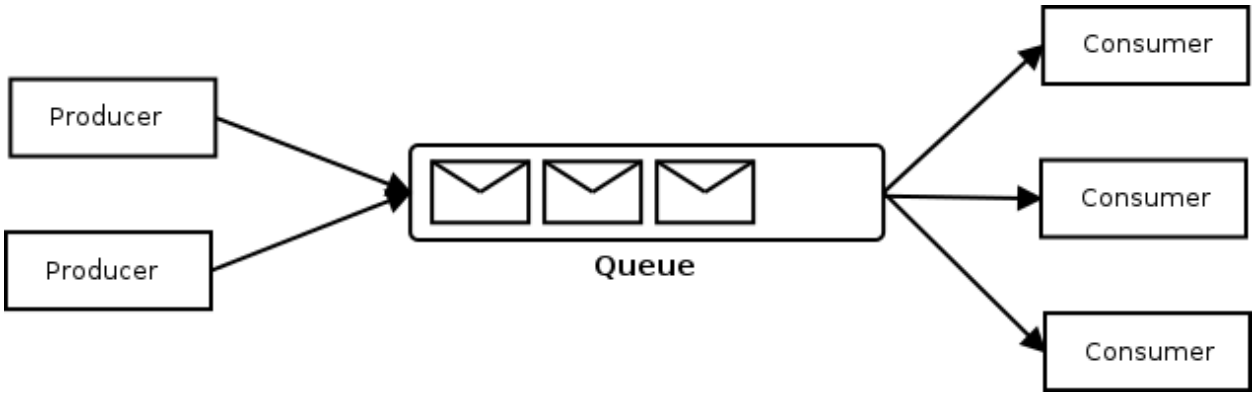
Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Spring Semester, 2022-23

A **distributed queue** is a type of data structure that is designed to hold and manage a large number of items or tasks in a distributed system.

It allows multiple processes to add and remove elements from the queue simultaneously, and ensures that tasks are processed in a **first-in-first-out (FIFO)** order across all nodes in the system. This helps to balance the workload and prevent bottlenecks in large-scale systems.



PART - A

We have implemented the distributed logging queue using **Python along with Django**, which is a high level python framework for supporting REST APIs.

Part A Design Decisions -

- All the log messages will be available to the new users as well i.e. even after all current consumers have viewed the message it is not removed from the queue. This gives access to old messages to all new consumers, which might be necessary.
- Class DistQueue is implemented to handle the log messages along with the producers and consumers.

- All registered producers and consumers are stored in a dictionary reg_prods and reg_cons, with topic name as key and the list of producer and consumer ids respectively registered to the topic as value
- All topics are also stored in a list, topics. The topic name is any non-empty string.
- The log message queue containing all log messages is also stored as a dictionary with the topic names as key and a tuple containing the producer id of the producer who created the message, the message text and the list of consumers who have viewed the message as value.
- Producer and Consumer ids are assigned sequentially to producers and consumers starting from 1.

PART - B

We have added **persistence** to the distributed queue, so that log messages are stored in a persistent storage layer and can be retrieved even if the server crashes or restarts.

Part B Design Decisions -

- All the log messages will be available to the new users as well(give some reasonable explanation).
- Database is implemented using PostGRESQL (currently SQLite).
- Django handles all the low level detail communication with the Database.
- There are multiple models(tables), they are Topic, Producer, Consumer, LogMessage mainly.
- There are two other linking tables which link consumers with their subscriptions (Consumer Subscriptions) and another table which links log messages to the consumers who viewed the messages (consumerViews).
- There are various fields like TextField, DateTimeField etc.
- Consumer and LogMessages are linked using a ManyToManyField .
- LogMessages are sorted based on their creation time order(by using the ordering = [‘created’] in the Meta Class of LogMessage).

- All the functions implemented in Part A have been implemented again in the file queue_funcs.py using the databases.

Hyperparameters: Various fields like textfields, datetimefield, manytomanyfields etc in various tables in the persistent storage. Topic names are restricted to 200 characters.

PART - C

We created a Python library which acts as an abstraction using which producers and consumers can communicate with our distributed logging queue using a more convenient or efficient interface.

Part C Design Decisions -

- A package is implemented containing the class myQueue, which provides an easy to use user interface for any user using this distributed logging queue system.
- The class stores the server link, and provides functions to users to create topics, list topics, register consumers, register producers, enqueue, dequeue and get the size of the queue.
- These functions take the parameters as inputs and call the necessary requests to the http server.

Testing

We made 5 producers and 3 consumers with 3 topics using the library developed in Part-C and tested them using on various producer and consumer mapping.

- Command to run test file python3 testing.py

Challenges faced

1. This is our first full - fledged project using Python and Django, which we have to learn before implementing this assignment.
2. Django uses multiple threads, as a result implementing Part A was a challenge, wherein we were not using any persistent storage layer which resulted in difficulties.